**MASTER IN COMPUTER SCIENCE**

# An experiment translation through automatic generation & a secure routing protocol implementation FUUREX

Van de Sype, Julien; Guillaume, Laurent

*Award date:*
2010

*Awarding institution:*
University of Namur

**Facultés Universitaires Notre-Dame de la Paix**
**Faculté d'Informatique**
**Second Master**

**Master thesis**

# An Experiment Translation Through Automatic Generation & A secure routing protocol implementation: FUUREX

Julien Van de Sype
Laurent Guillaume

**Academic year** 2009-2010
**Supervisors:** Laurent Schumacher - Roberto Canonico

---

**Faculté d'Informatique - rue Grandgagnage, 21**
**B-5000 Namur, Belgique**
**Tél : +32 (0)81 72 50 02 - Fax : +32 (0)81 72 49 67**

**Facultés Universitaires Notre-Dame de la Paix**
**Faculté d'Informatique**
**Second Master**

---

# Master thesis

**An Experiment Translation Through Automatic Generation**
&
**A secure routing protocol implementation: FUUREX**

**Julien Van de Sype** - jvdsype@student.fundp.ac.be
**Laurent Guillaume** - guillala@student.fundp.ac.be

**Academic year** 2009-2010
**Supervisors:** Laurent Schumacher - Roberto Canonico

---

**Faculté d'Informatique - rue Grandgagnage, 21**
**B-5000 Namur, Belgique**
**Tél : +32 (0)81 72 50 02 - Fax : +32 (0)81 72 49 67**

# Abstract

Computer investigation in the scientific community often leads to the design of new theories and their sometimes empirical validation. In order to save time and because of the lack of available equipment, researchers compare their findings to a simulation from which they will draw conclusions about the results. Unfortunately these tools are more or less far from reality. Implementations coded by researchers should therefore also be tested by other means closer to reality.

On one hand, this master thesis presents the reader with an approach to facilitate the joint use of a simulator and a testbed in the context of testing software involving wireless networks. On the other hand, it describes the novel security brought by the AODV routing protocol augmented with a mechanism of reputation.

We propose the development of a tool based on the XML language to automatically generate descriptions of common experiments between the simulator and the testbed. The second part focuses on the implementation of a secure routing protocol through a joint collaboration between the participating nodes in the wireless network.

Our platform currently manages the NS2 simulator and an OMF testbed, wherein we have also enabled the exploitation of the modified routing protocol. Moreover, the range of tools considered can be expanded easily.

Researchers now use a solution jointly operating two tools which help save time when testing applications and as well as secure implementation of the AODV protocol.

**Keywords**: XML, testbed, OMF, NS2, AODV, Wireless Mesh Network

# Résumé

Les recherches informatiques dans le milieu scientifique donnent souvent lieu à la création de nouvelles théories ainsi qu'à leur validation, quelque fois empirique. Pour un gain de temps et par manque de matériel disponible, les chercheurs confrontent leurs trouvailles à un outil de simulation dont ils tireront des conclusions sur des résultats obtenus. Malheureusement ces outils étant plus ou moins loin de la réalité, les implémentations des chercheurs devraient être également testées par d'autres moyens plus proches de la réalité.

Nous proposons le développement d'un outil se basant sur le langage XML afin de générer automatiquement des descriptions d'expérimentations communes aux simulateur et aux testbeds. La seconde partie s'intéresse à l'implémentation de la sécurisation du protocole de routage AODV grâce à une collaboration commune entre noeuds participants au réseau sans fil.

Notre plate-forme gère actuellement le simulateur NS2 ainsi que le testbed OMF, dans laquelle nous avons également rendu possible l'exploitation du protocole modifié. De plus, la gamme d'outils prise en compte peut être étoffée aisément.

Les chercheurs disposent maintenant d'une solution exploitant conjointement les deux outils permettant un gain de temps pour le test d'applications ainsi que d'une implémentation sécurisée du protocole AODV.

**Mots clés**: XML, testbed, OMF, NS2, AODV, réseau maillé sans-fil

# Acknowledgement

We would like to express our deepest and sincere gratitude to our thesis supervisor, Professor Laurent Schumacher for all the advice, the corrections, enlightenments and support we received throughout this year. We are really glad and pleased he offered his guidance and his network orientation knowledge.

We gratefully acknowledge Professor Roberto Canonico for the supervision and the help he brought us during these three months in Naples. We would like also to underline his never ending enthusiasm on the XML project.

We would also wish to record our gratitude to Giovanni Di Stasi, the PhD Student who backed us up taking his time to answer our numerous questions and providing us with a crucial technical support.

We would like to thank Francesco Oliviero for his researches in wireless routing protocol security which made our security approach in our thesis possible.

Our gratitude also goes to Professors Wim Vanhoof and Patrick Heymans for their help in the numerous tricky problems we encountered. They helped us tackle them, and get a better overview on the subject.

Next we would like to thank Alessio Botta and Francesco Paolo for their enlightenments during the research period.

Finally, our regards and blessing go to all those who supported us in any respect during the completion of the project.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ANML** Another Modeling Language
**AODV** Ad-hoc On-Demand Distance Vector routing
**AODV-UU** Ad-hoc On-Demand Distance Vector routing - Uppsala University
**AS** Autonomous System
**CBR** Constant Bit Rate
**CIA** Confidentiality, Integrity, Availability
**CONFIDANT** Cooperation Of Nodes: Fairness In Dynamic Ad-hoc NeTworks
**DITG** Distributed Internet Traffic Generator
**DML** Data Manipulation Language
**DSDV** Destination-Sequenced Distance Vector routing
**DSL** Digital Subscriber Line
**DSR** Dynamic Source Routing
**DTD** Document Type Definition
**EBNF** Extended Backus-Naur Form
**FTP** File Transfert protocol
**FUUREX** Fundp Uppsala University REputation eXtenstion
**GMF** Graphical Modeling Framework
**GML** Graphic Modeling Language
**GUI** Graphic User Interface
**HTML** Hypertext Markup Language
**MAC** Media Access Control
**OEDL** OMF Experiment Description Language
**OLSR** Optimized Link State Routing protocol
**OMF** cOntrol and Management Framework
**oTCL** object-Tool Command Language
**OTG** Orbit Traffic Generator
**NS** Network Simulator
**REFACING** RElationship-FAmiliarity-Confidence-INteGrity
**RERR** Route ERRor
**RREQ** Route REQuest
**RREP** Route REPly
**RREP-ACK** Route REPly ACKnowledge
**RTF** Rich Text Format
**SQ3** Space Quest 3
**TCL** Tool Command Language
**TCP** Transmission Control Protocol
**TTL** Time To Live
**UDP** User Data Protocol
**UML** Unified Modelling Language
**W3C** World Wide Web Consortium
**WMN** Wireless Mesh Network
**XDLWNS** XML Description Language for Web−based Network Simulation
**WILE-E** WIreLEss Experimental
**XML** eXtended Markup Language
**XSD** XML Schema Definition

**XSL** Extensible Stylesheet Language
**XSLT** Extensible Stylesheet Language Transformations

# Chapter 1

# Introduction

The cost in the domain of the network development has nowadays become a real problem. In order to perform wide scale experiments, a large number of hosts, routers, switches,... is required. But many companies or even universities do not want to invest much money into hardware that becomes quickly obsolete. The simulation happens to be a solution to that problem. People are able to virtually create their wide-scale scenario and perform their experiment at a way lower cost than with a conventional hardware-based set-up. However, this virtual network has the drawback of missing the effects that could affect the experiment in real conditions. Although some disturbances can be introduced in the scenario, those experiments are not as realistic as the real ones. The testbeds should overcome this issue. A testbed is a set of computers gathered in order to perform experiments. That hardware enables to carry out the experiment in realistic conditions under planned restrictions.

These tools thus act on two different levels, one in the real world and the other in the virtual world. Testbeds and simulators are complementary since the weaknesses of one are counterbalanced by the other's strength.

In practice, developers often use the simulators. Simulators are of better value in terms of cost and quality of results they are able to provide. Even if it was wise to combine simulators and testbeds, such a pairing would often require an additional effort from the developers. They will have to write a second implementation to be run on the testbed and would secondly require the availability of such tools. Some projects are attempting to overcome the availability problem by allowing outside access to their testbed for developers from all around the world.

The test of these applications using both tools is made by running a scenario created by the user in which s/he has depicted a real world situation that s/he would like to reproduce. When a researcher wants to take advantage of the benefits offered by the tools combination to test their software (compare / validate the results), s/he should configure them separately although they follow the same scenario.

It is in this context that we address the first part of the thesis, consisting in finding a way to facilitate the management of the same scenario on both platforms. This first goal is detailed in the following section.

With this in mind, we tested this approach with a real case involving the security of wireless communications.

More and more companies and industries are now building and creating their business with their customers and create links with new potential customers through the Internet bubble. Residential customers are also deploying wireless networks in order to provide to all family members with an Internet connection from a single access point. Universities are based on the same pattern of wireless use as residential customers. Because of the swarm of new users, the numbers of services available are increasing, which makes this type of market a thriving business.

Nevertheless, the side effect of this technology is the technology itself. The physical support for propagating packets is based on airwaves resulting in a weakening of these networks' security compared to wired links, especially as the airwaves can be intercepted and sniffed by anyone within range.

The security problem can be exploited to lower quality of service through the interception of messages and attacks. Preventing a company from providing services to its users is not only limited to a financial loss and but it can go further by directly affecting the image of company, reflecting the confidence of

customers towards it.

It is in this context that we implemented and tested a security solution to counter the actions of malicious network devices in a wireless network. This point is discussed in the next section.

## 1.1 Objectives

### 1.1.1 Automatic experiment generation

The first goal is to create a generic language to describe a scenario of network experiment. Once this description is done, the scenario described is processed in two different ways to be instantiated for a simulator and a testbed.

Both tools are platforms used to test softwares, routing protocols, algorithms, etc. Indeed, when researchers are developing computer programs, they need to be tested in laboratories to be used in production once validated.

Therefore it is interesting to collect and verify the data they produce in order to draw conclusions. The first tool is a simulator system, i.e., a software that virtualizes the whole test environment: devices and communications are virtual. The second tool is a testbed, i.e., several real devices (unlike the simulator) communicating with each other and managed by a software playing the role of a controller.

These two tools are therefore intended to perform an experiment to collect data and to provide feedback to the user so that s/he could study them. As discussed in the previous section, one is virtual while the other acts in the real world. They are thus complementary.

By describing a scenario of a networking experiment, the user wants to test his/her application (for example, a routing algorithm). S/he then designs a scenario in which a specified number of devices (called "nodes" in a network topology), which may or not communicate with each other. It also specifies, for example, that node A will generate traffic to node D via node B, etc. Thus, this scenario determines the parameters that each device will receive.

However, both tools use two different languages to describe an experimental scenario. So, the first aim of this thesis is to find a fairly generic way to express this scenario only once and only for the wireless part of a network topology, as well as to provide an automatic transformation tool.

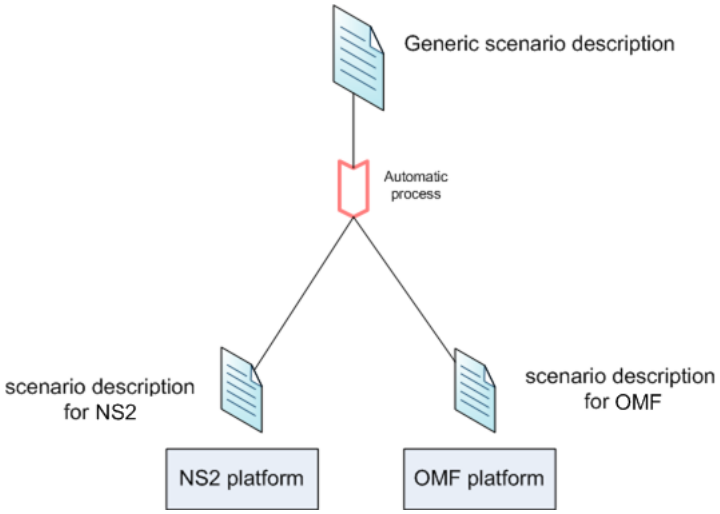



Figure 1.1: The first objective is to provide a generic description language and an automatic transformation tool

Using various tools, the generic description is transformed once in the language for the simulator and once in the language for the testbed, as shown in Figure 1.1.

To sum up, our first objective is to define a common description language for network experiments to automatically produce description files for simulators and testbed platforms.

### 1.1.2 Implementation of a reputation protocol

Because the need in wireless network is now a bigger problem, mechanisms should be deployed to secure communication. As the entities communicate over the air, any malicious node can intercept the conversation and act in a deviant way, so the exchange between nodes is either wrong or broken. To prevent such situation from happening, new modules and theories have been developed. We will implement one of them, called RElationship-FAmiliarity-Confidence-INteGrity (REFACING), on a well-known wireless routing protocol named Ad-hoc On-Demand Distance Vector routing (AODV).

Starting from the AODV source code, we are going to add the required features to embed the REFACING model and it will therefore be possible to test the modified routing protocol on a platform.

### 1.1.3 Thesis contents

This thesis is organized in two parts. The first one which deals with the automatic generation tool, presents the situation by a *state of the art* at chapter 2. Chapter 3 introduces the components of the simulator exploited in this project, while chapter 4 focuses on describing the testbed plaform we got to use. Chapter 5 discusses the existing researches and the solution we propose. Chapter 6, the last chapter of the first part, addresses the limits and perspectives of our work.

The first chapter of the second part, concerning the security mechanism in a wireless routing protocol, introduces the concepts of wireless networks and lays the foundations of the security issues in those ones. Chapter 8 describes the operating principles of an existing insecure wireless routing protocol as well as the means to make it secure in accordance with a new theory. Chapter 9 looks at the implementation of the protocol used in this project. A list of comments takes place in this chapter as well as an analysis of the theory limits explained in the previous chapter. Experimentations on our implementation and the results we got are expressed in Chapter 10. We finish this second by an analysis of the implementation limits and the solutions we proposed in regards to them. Finally chapter 12 is devoted to our thesis conclusion.

# Part I

# Automatic generation

# Chapter 2

# State of Art

This chapter introduces the notions and concepts in the context of the first objective we have set. It describes the existing tools and the environment in the context of our objective of generating a network experiment in an automatic way.

## 2.1 Presentation of simulators, emulators and testbeds

Tests of network applications can be achieved in three different ways :

- Simulators
- Emulators
- Testbeds

Every single of these three methods can perform the application and thus test and check its behavior in different conditions. Let us define and consider the advantages and disadvantages of them.

## 2.2 Simulators

The simulators are part of an experimental work. We will therefore analyse what simulators are and what they are used for.

**Simulator** A simulator is a system to mimic the behavior of a set of machines in order to run experiments in a given context. It is often used for the study of real-life complex systems for which the all combined parameters are too complicated or too expensive to be run in normal scale. A model is therefore made to be as close as possible to reality, so the analysis is as real as can be. Nowadays, simulators in computer science are used to check the behavior of new designed softwares or theories. It allows to test them in an environment that tends to be as realistic as possible.

### 2.2.1 Network Simulator

The Network Simulator (NS) is the most famous simulator in its domain, as shown in [AY06]. It was created as a division of the original *REAL network simulator* in 1989. The software then underwent several changes to adapt through work of researchers. The second release, also known as NS2, was distributed in 2005. This is currently the most used version, mainly thanks to its combination of Tool Command Language (TCL) language and commands, and its C++ capability and appliance. The last version so far was released in 2009 under the name NS3. It lost its TCL part, but major changes have been done to fix problems in the NS2 version.

In our thesis, we will only focus on the second version of NS. This release had three great changes that are important to consider.
The first one was incorporated in the very first version of the software: the wired links between nodes.

So, all nodes are then set together thanks to physical connection. The second change was the one made by the University of Cantabria [CC07]: the wireless links to build up the topology. But only the 802.11x standard has been implemented. And the last change was done by Francesco Oliviero at the University Federico II of Naples [OR07], who added a reputation protocol - that will be discussed in a further section - to identify and remove a node that behaves suspiciously and whose fishy behavior makes the network efficiency going down.

### 2.2.2 The other simulators

Other simulators that we could use instead of NS exists. Here is a short list of them:

- Marionnet
- Netkit
- SNNS (Stuttgart Neural Network Simulator)
- GNS3 (Graphical Network Simulator)
- JNS (Java Network Simulator – Clone of NS)

All are simulators able to graphically model a network with a single machine at a lower cost. These are great and well-known simulators. However, none of them includes the wireless part of a network, as they are all designed to model wired networks. And because of that lack of evolution and maintenance, they currently cannot keep up with the NS functionalities.

## 2.3 Testbeds

In this whole thesis, we will often address and use the notion of testbed. Let us define and explore what *testbed* refers to.

**Testbed** A testbed is a set of managed experimental resources (often computers or small wireless devices), which can be configured and controlled by experimenters/researchers for a specific period of time, to perform experimental evaluations of algorithms, schemes or prototypes. Testbeds enable the realization of experiments that are often run on real-world scenarios, under controlled realistic environment. It consists of specific hardware, operating systems, network infrastructure and configuration [cO09b].

A testbed can be viewed as different devices owning several network interfaces, often managed by a central controller which provides help and tools to run experiments on these devices in a very simple way. It proves useful in the test of routing protocol implementation, in the test of applications by ensuring conditions fairly close to reality, unlike simulators.

Testbeds can be divided in two categories, the local and the global scale ones. The local testbeds are located in a single area in the world, like a room, a laboratory or a University whereas the global testbeds have nodes spread all around the world. The classification does not depend on the number of nodes present in the testbed, but on the location it actually has. [cO09a]

### 2.3.1 PlanetLab

PlanetLab [Pla09] is a worldwide testbed to which many countries contribute in adding their nodes in that topology.

It is a global research network providing a large scale infrastructure for the development of new network services. Since 2003, more than 1,000 researchers from industrial and academic world have used the testbed for developing applications such as peer-to-peer systems, distributed storage, distributed hash tables, network mapping and query processing.

Each research project is assigned into a slice, i.e., a virtual access to some of the nodes in the network. PlanetLab currently consists of 1,086 nodes at 505 sites all around the world.

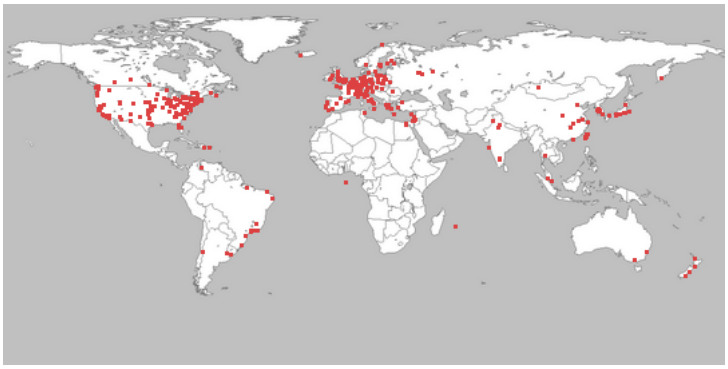


Figure 2.1: PlanetLab organisation: 1,086 nodes at 505 sites

To join the PlanetLab testbed, the institution responsible for the application must have at least one partner which is already member of the PlanetLab consortium.

### 2.3.2 WILE-E

A prototype testbed called WILE-E is deployed in a laboratory in the University Federico II of Naples. It is composed of seven wireless nodes, managed by a testbed controller. The nodes internet access is provided through a gateway and PlanetLab developers can connect to the testbed manager to perform experiments.

The four older nodes NETGEAR WG302 have 32 MB of memory RAM. Those nodes embed a Linux based operating system called OpenWrt (Busybox v1.14.4). However, even if they have two antennas, the first one is only dedicated to the 802.11a while the second can only be used for the 802.11g protocol.


Figure 2.2: Netgear WG302 and net4826 SOEKRIS

The new generation is composed of three nodes, the net4826 SOEKRIS Engineering with 128 MB of memory RAM. This generation embeds a Linux operating system called Voyage (version 4.0).

Currently deployed in the University Federico II of Naples, the testbed is composed of the nodes described above, a gateway which provides Internet and a server acting simultaneously as a manager for the testbed (see figure 2.3) and as PlanetLab (see 2.3.1) node for external researchers who want access to this infrastructure.

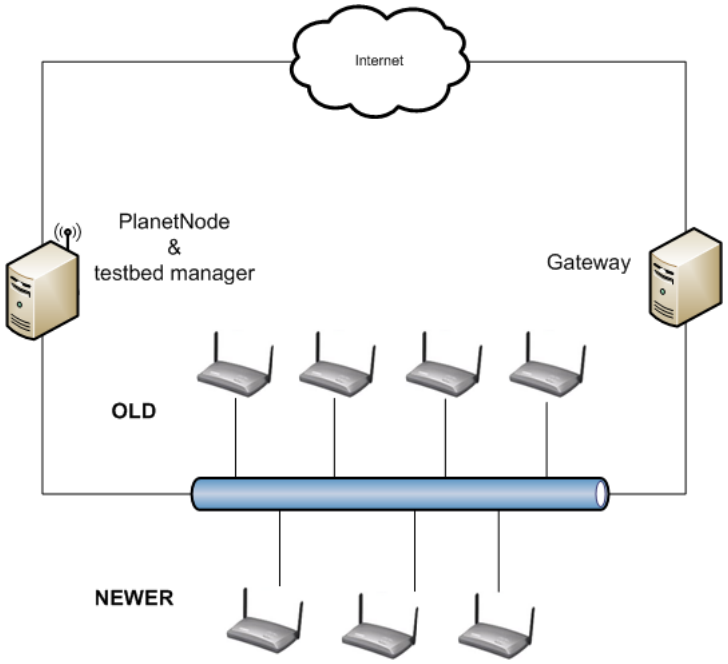



Figure 2.3: WILE-E infrastructure

### 2.3.3 Emulab

EmuLab [Emu09] is part of global testbeds and is a really similar project to the one made by PlanetLab.

Although the Emulab testbed is spread all around the world, the core system is located at the University of Utah, U.S. (with around four hundred machines). Two thousand sites compiling an average of 2,500 nodes and machines can be added to this core.

This testbed is considered as a *facility* and a *software system*. It provides a high number of nodes to run experiments, but also some emulation and simulation stuffs to couple the conditions of testing. That advantage is thus twofold: allowing users to make and study their trials on different platforms under specific conditions and at different places.

The specificity of this testbed is the access to a large number of experimental environments for the different needs a user could have. Those environments are :

- Emulation environment
- Resilient Overlay Networks and PlanetLab synergies
- 802.11a/b/g standard
- Software-defined radio
- Motes within network
- Simulation environment

These environments make Emulab one of the most user favourite testbeds.

## 2.4 Testbeds management framework

### 2.4.1 OMF

The cOntrol and Management Framework (OMF) is a set of software tools to control, measure and manage networking testbeds. This software system can be used with several heterogeneous systems, each one consisting of different kinds of devices. That is why this software suite, acting as a platform, is deployed on testbeds using different kind of hardware, in Australia, Europe, and in the U.S.

Initially, OMF was developed for the Orbit wireless testbed, a radio grid that is currently being developed for the evaluation of next-generation wireless network protocols [OL09]. Since 2007, the OMF capacity has been extended to consider a wider range of testbeds using different types of networks. These modifications have made OMF to become an open source framework, kept regularly updated by taking into account new technologies providing support for wired and wireless networks.

The Federico testbed (WILE-E) is managed by this framework. OMF will be described in details in the dedicated chapter 4.

### 2.4.2 Castadiva

Castadiva [JHM09], which provides an open-source management tool, is also a testbed composed of two piles of routers, 21 Linksys routers in total. The operating system of Castadiva is a Linux distribution, named Open-WRT, running on wireless nodes. This testbed is presented as two stacks of routers that are piled up. The project is coupled with a simulator, namely NS2, as it is also considered by the creators to be the widest used software of its category. The testbed is made up so the sole experiment description is written in TCL and the interpreter will then browse the file and will be able to run both the testbed and the NS2 simulator from that single point of input. This is an example of what this thesis wants to achieve.

The user interface of Castadiva can handle the experiments to be run on the testbed. That Graphic User Interface (GUI) can control all the testbed features, from defining the topology to setting all the flows parameters. It allows to define all the parameters step-by-step, and even see the simulation to be run in real time, like it could be with the NS2 animator *nam*. That user interface is an open-source GUI, which can be found at `http://castadiva.sourceforge.net`. This testbed is a local Spanish testbed.

This project is the closest to the one we will present in our thesis. However, it is important to notice that the base language is different, which is probably the biggest difference in the purpose of our respective work (eXtended Markup Language (XML) instead of TCL).

## 2.5 Pros and cons

Considering the simulation, it is probably the cheapest way to carry out experiments at low cost. It is also easier to run complex scenarios which can include a great number and/or a great variety of devices. In the sole case where both the simulation parameters and the models used are correct, the simulation can be considered as *realistic*.

In order to analyse a protocol (or something else), it is not possible to use its original implementation in the simulator. This is why developers have to create another implementation, which is quite different from the original one. Moreover, the cost of certain operations from the real work is not easily quantifiable. It is so either because too many factors are coming in the frame, or because the behavior is hard to model and all those elements are thus difficult to reproduce.

It is difficult to know if the models, a reduction and an abstraction of reality, do ignore certain important aspects of reality. Therefore, for all of these reasons, we noticed that we should not only rely on a simulator as a proof to validate the good working of the software.

An emulator is an hybrid environment coupling both simulation and real-world environment. The applications and the protocols are using a real implementation whereas the rest is simulated. It is a good compromise to reduce the gap with the simulation-related reality.

On the other side with testbeds, it is possible to use realistic parameters without any difficulties. The running of the given network application can be done in an environment closer to reality and the should-be conditions. The cost of operations is no more a problem, even if it is dependent on the hardware and can be measured easily. It is still not the real world but we are getting closer to a realistic behavior.

In counterpart, testbeds happen to be more expensive, harder to configure and to manage. To these disadvantages, we can add the scaling limit of the testbed for which an experiment involving hundreds of nodes would be almost impossible. An experiment including only a few nodes can not be representative of reality.

For the reasons mentioned above, it is meaningful to use a combination of both tools and we believe that this approach should be conducted during research works.

## 2.6 Languages used by testbeds and simulators

### 2.6.1 OMF language

A Domain-specific Language [AvD10] is employed in OMF to describe an experiment and has been called OMF Experiment Description Language (OEDL).

Ruby language [Rub10] is the basis of OEDL on which some statements and commands needed in OMF have been added. Since this interpreted and oriented object language is quite 'human-readable', the grasp of Ruby is easier and does not require much learning from the user to be able to write an experiment.

### 2.6.2 Network Simulator 2 language

The network simulator we are using in our experiment scenarios has been mostly written in C++, and some modules in TCL. All its modules and features required those two languages. However, the experiment files required for input are based on the TCL language, and C++ features are added for an easier code.

It enables easy-written experiments, and that is one reason why the translation from XML to an experiment file targeted to NS2 was possible without complex or irresolvable problems.

# Chapter 3

# Network Simulator

This chapter introduces the components used in the NS2 simulator, explains our contribution and the perspectives in this field.

## 3.1 Introduction

In the network development, it is sometimes desirable to get an accurate idea on how our designed topology can behave using different protocols in a particular environment. Some tools are available to run a virtual experiment helping us to record some information about the network flows and behavior. One of those tools is NS2, a network simulator. A network simulator is a software able to virtually create a set of machines and data flows between them according to a specification received in input in order to test internet protocols. For instance, NS2 uses TCL-based files (pronounced "*Tickel*") describing the experiment and all the elements to properly build a network. We can define the links between a sender and receiver along with their bandwidth, rate and packet size. Then we can specify the machines protocols, the numbers of channels (in wireless) as well as the traffic model and its specifications.

## 3.2 The components of NS2

In the previous chapter, we introduced the fact that NS2 is a network simulator, and was written in the TCL programming language. But the major part of the network simulator is written in C++. Those two languages are used to define the different modules of the simulator.

Basically, the NS simulator software has three major parts that compose it and its extensions.

The first part is the routing protocol implementation. The routing protocol implementation is a part of the whole simulator, but has its own definition in it. It is then possible to define several routing protocols (the basic one being AODV, but it is also feasible to declare the Optimized Link State Routing protocol (OLSR) or the Destination-Sequenced Distance Vector routing (DSDV) routing protocol) with each of them having their own directory. So, the routing protocol has its own part in the simulation organisation. The routing protocol is implemented in C++, as well as the whole main simulator part.

The second part is the experiment description. The simulator allows to describe the experiments in object-Tool Command Language (oTCL), an object-oriented language (Table 3.1 illustrates the required TCL code to run an experiment on the NS platform). When NS2 is interpreting the TCL files, it launches an event scheduler. This event scheduler is responsible for the traffic time handling as well as the time to begin and finish the events and when to start and stop the experiment.

The link between those two different languages is made by TCL, which will make the linking and translation from the TCL language (and the experiment objects) to the C++ one as shown on Figure 3.1. This translation is mainly made to reduce the interpretation time of the simulation.

| TCL code | Meaning |
|---|---|
| Flows definition | |
| set trSource(a) X,Y | Defines the source node flow "a" with position (X,Y) |
| set trSink(a) X,Y | Defines the sink node flow a with node position (X,Y) |
| Traffic definition | |
| set sink(k) [new Agent/Loss-Monitor] | Declares the $k^{th}$ sink according to a sink model |
| $ns_ attach−agent $node_($trSink(k)) $sink(k) | Attaches the $k^{th}$ sink function to the $k^{th}$ sink node |
| set sourc(k) *Traffic* | Affects the *Traffic* definition to the $k^{th}$ source node |
| Nodes definition | |
| $ns_ change−numifs XY | Initializes the number of channel for the node to XY |
| $ns_ add−channel XZ $chan_(Y) | Adds the channel number XZ to be at value Y |
| set node_(X,Y) [$ns_ node] | Defines a node with position values (X,Y) |
| $node_(X,Y) set X_ A | Gives the value A to the coordinate X |
| Nodes definition | |
| $ns_ at s,t $sourc(k) {start,stop} | {Start,Stop} the traffic source numbered k at second(s) s,t |
| $ns_ at s,t record $sink(k) $f(l) | Start logging the received bandwidth at second(s) s,t of $k^{th}$ sink in the $l^{th}$ log file |

Table 3.1: TCL NS-specific code samples

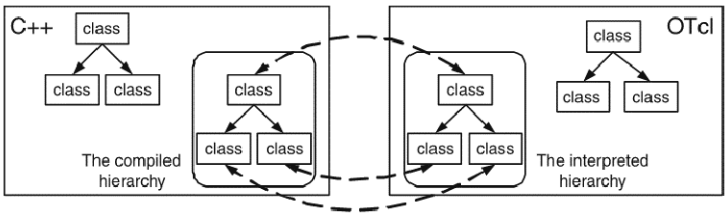



Figure 3.1: The linking between C++ and oTCL

Finally, the results from the compilation by the simulator can be read by the *nam* tool, the network animator. It gives a graphic representation of the topology and scenario of the experiment.

## 3.3 Why NS2 ?

The choice of the second version of the simulator in our work instead of the third version is partly based on arguments from the University Federico II. The main reason is because of the changes made on the NS experiment files. The third version enables only C++ files in input whereas the NS2 one allows TCL experiment descriptions. The removal of the TCL ability represents a loss for the current user. The TCL is an interpreted language. We can therefore modify some parameters in real-time during the experiment whereas it cannot be done in NS3, because the files have to be compiled. It is thus easier to use the TCL implementation to add new possibilities and commands to the framework. Besides, TCL has been basically created for defining rapid scripts and easy to use applications, which is what is actually sought for. Moreover, the NS3 version is only based on the C++ language, which implies more knowledge and complex experiment description than the previous version.

In addition, the add-on to enable the normal wireless network as well as the wireless mesh network on NS2 makes it a must in the domain of simulators and a reference for both wired and wireless network.

It is also important to notice that many other works similar to ours, have also considered NS2 for the translation to a simulator script. This strengthened our mind on the choice of NS2 as a good target for a simulator, which was imposed anyway.

Professor Canonico, our advisor in Naples, told us they are currently using NS2 for their simulations. The University basically prefers to keep that version instead of using the new one and making their old scripts and way of writing experiment obsolete. Our second motivation to use NS2 was the fact that we were starting from the work of Prof. Canonico, a work made six years ago for the same simulator. The XML project started at that time already had its set of XML Schema Definition (XSD) descriptions and all the elements described were all defined properly at that time (a research paper has also been published [RCG03]).

For all those reasons, we opted for NS2 in the simulator part of this research.

## 3.4 Related contribution

Related to the NS simulator, we contributed by adding wireless network elements over the already existing wired elements in the XML / XSD / Extensible Stylesheet Language (XSL) document.

As we will detail it in the following chapter, we created an automatic translation module to convert the generic experiment description written in XML into two platform-specific files, namely OMF and NS2. This transformation makes the translation for the wireless part over the existing implemented wired module possible. It allows from a single experiment description file to obtain two NS-specific environments that can be run either in a wired network or in the wireless one.

## 3.5 NS Future

After investigations, we found out that a group of researchers came up with a similar idea of an automatic translation for the NS3 version, as detailed in [HWP09]. Even if NS2 is widely used for now, with this new easier way of describing an experiment for the third version of NS, we can suggest that the newest

version will become the new reference. The old NS2-specific files will just have to be re-encoded to pass to the new release, once the wireless is enabled.

# Chapter 4

# The OMF Platform

This chapter presents the tool we mastered to conduct experiments in the real-world network. First, it presents a brief history of OMF and the utility of such a system for the software validation. It also discusses the working of the tool and illustrates it with an example, in modelling the different concepts of OMF. Then the software version used is justified and we show how the database stores information about equipment. Finally, our contribution is introduced.

## 4.1 Introduction

OMF (cOntrol and Management Framework) is a testbed cOntrol, Measurement and Management Framework. In other words, OMF is a set of tools to control, measure and manage networking testbeds. This controller has the function to setup all the devices belonging to the experiments, to carry out experiments and to gather measurements during the experiment.

Initially, OMF was developed for the Orbit (Open Access Research Testbed for Next-Generation Wireless Networks) wireless testbed. Orbit is a radio grid that is currently being developed for evaluation of next-generation wireless network protocols [OL09]. But since 2007, a lot of developments have been performed and extensions have been added in OMF. These modifications have become an open source framework which provides support for wired and wireless networks.

Currently, this framework can be used with several heterogeneous systems, with a lot of devices. That is why it is deployed and used on different testbeds in Australia, Europe, and in the U.S.

## 4.2 Why OMF ?

OMF aims at providing a means of executing experimentations in a real and controlled environment. New networking technologies are developed thanks to several different factors such as new discoveries in scientific fields, by the expansion of Internet or with the creation of new end-user services (Video-On-Demand).

Therefore, we need to evaluate the new networking technologies before they go out of the laboratories and are included in a commercial product, which may be brought to market. By evaluating a new technology, researchers want to check the behaviour of the new produced part under specific conditions to ensure that everything is working as it should.

As explained in Section 2.1, researchers have three kinds of relevant solutions to evaluate the behaviour of the future product, respectively named simulators, emulators and testbeds.

The advantages of the latter can not be ignored because they offer some features not available in the others. Implemented applications in a real and controlled environment will provide results really close to reality. Thus, describing and performing a network experiment in a testbed is the best solution in terms of realistic behavior. The original software implementation can be exploited and no simplifying model of the reality is used.

However, testbeds tend to be harder to configure and manage. Yet, they are still a simplification as their size is usually smaller than a real deployment (less than hundred nodes). Such a testing in a real deployment would require an additional cost.

Being given that many testbeds are built only for a specific project with limited lifespan (they are dismantled by the end of the project), the use of long-term dedicated testbeds (not linked to a specific project but rather to an organisation) enables network developers to exploit it for a long period, which avoids wasting money and resources.

Finally, OMF is an active open-source project which offers the functionalities that we can expect from a testbed management tool. Considering the possibilities of maintaining the framework, the developer team is available and can be reached without difficulty, which is a great advantage in case of trouble with the configuration.

This open-source framework is currently exploited at the University Federico II to manage the WILE-E testbed.

## 4.3 OMF components

Figure 4.1 presents the main components of the OMF architecture [cO09a].

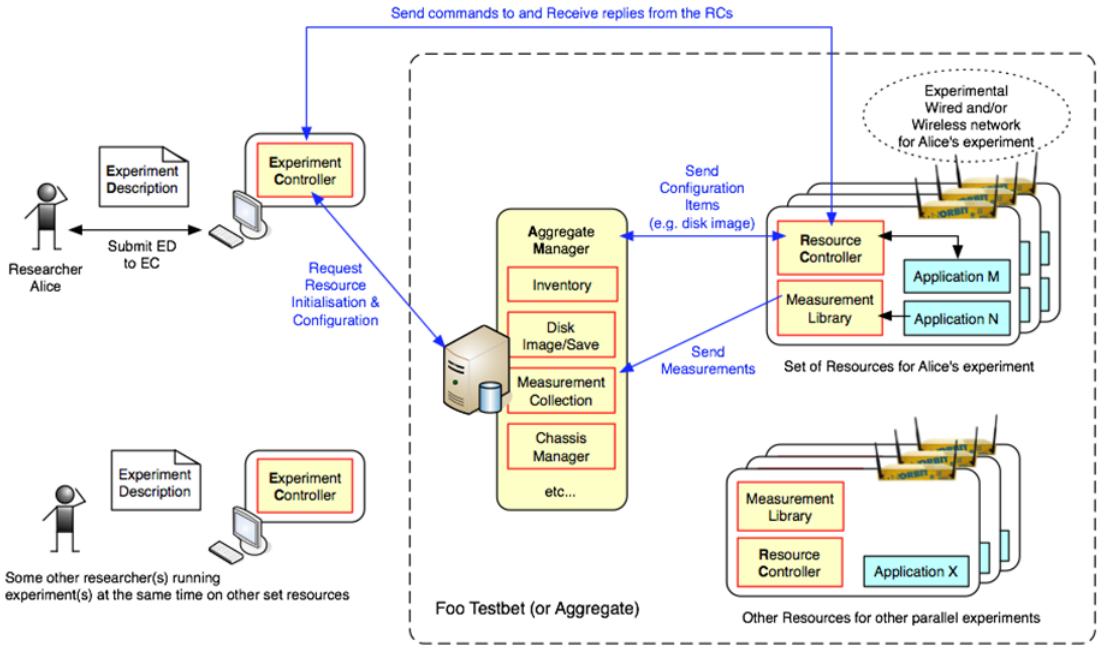



Figure 4.1: System Architecture from a User's Perspective Basic functionalities

### 4.3.1 Node handler aka the 'Experiment Controller'

The node handler acts as a director for the testbed. It is the entity in charge of deploying, controlling and executing an experiment on behalf of the user who describes it in OEDL (high-level language for OMF). It cooperates with the aggregate manager to configure the resources needed by an experiment. Once the resources are correctly set up, the node handler can communicate with the Resource Controller present on each node and send the commands to realize the experiment.

### 4.3.2 Node agent aka 'Resource Controller'

The node agent, present on each node of a testbed, listens to commands coming from the node handler, executes them and returns results to it. The node agent is also used to perform some management tasks on the node like installing a new disk image available on the grid service.

### 4.3.3 Grid service aka 'Aggregate Manager'

The grid service manages all the resources belonging to a testbed. It stores the disk images that can be loaded on the nodes. The grid service has the capacity to shut-down nodes, to boot them, to provide information about the inventory of the available resources, to give remote access to the database, etc. The database stores information about the nodes hardware such as Media Access Control (MAC) address, location, etc. in order to configure them correctly for an experiment.

## 4.4 Process an OMF experiment

Here is an activity diagram illustrating the processing of an OMF-experiment:

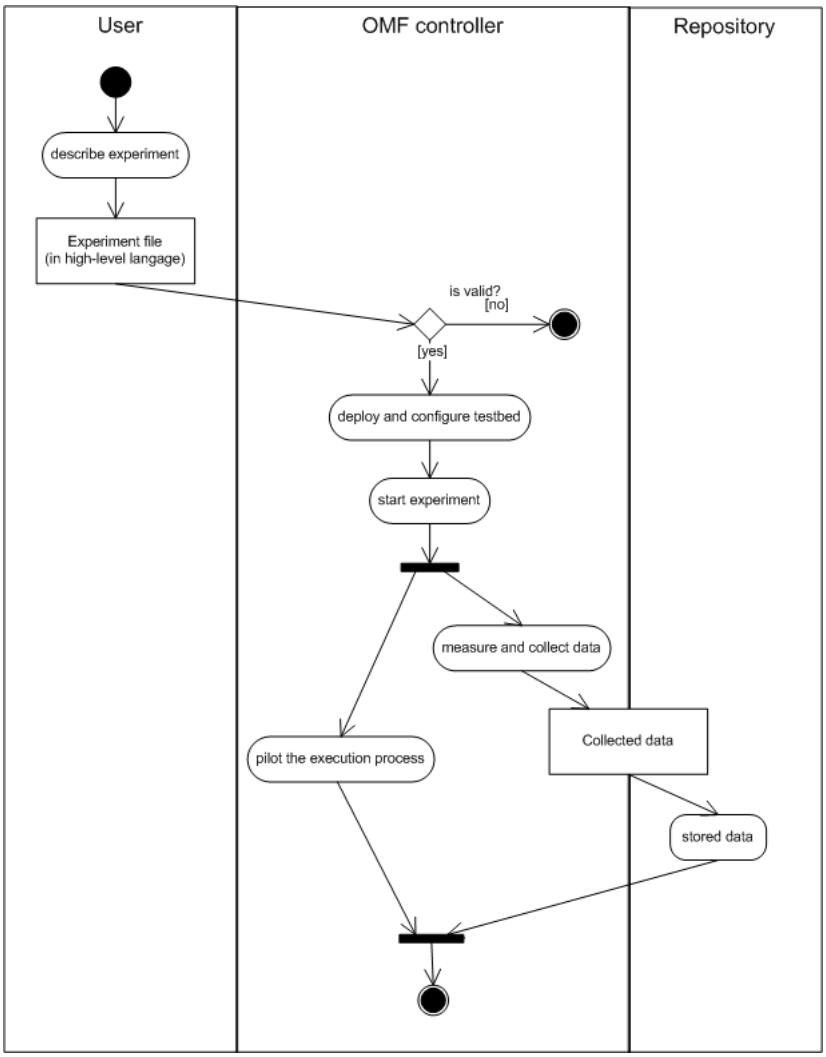



Figure 4.2: OMF process

Activity *measure and collect* data is carried out during the whole experiment, not only once. Then the experiment controller can take decisions based on the "on-the-fly" data, such as reducing the bit rate of a network interface if the error rate is increasing too much.

## 4.5 Description of an OMF experiment

This section is dedicated to explain to the reader a basic experiment on the testbed platform, which is the kind of description our tool will create. The experiment we chose as reference example is **The basic**

**"Hello World"** [cO10a].

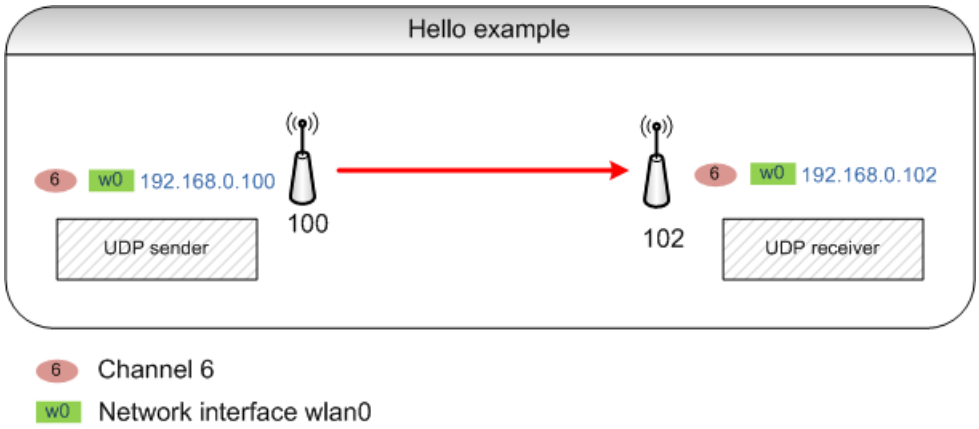



Figure 4.3: HelloWorld experiment

**Developing the "Hello World" Experiment Description**

```
#
# A) Define the 'sender' group, which has the unique
#    node [1,1]. Nodes in this group will execute the
#    application 'test:proto:sender'
#
defGroup('source', [8,100]) {|node|
    node.prototype("test:proto:udp_sender", {
        'destinationHost' => '192.168.0.102',
        'localHost' => '192.168.0.100',
        'packetSize' => 256,
        'rate' => 8192
    })
    node.net.w0.mode = "master"
}
```

This first section of the experiment defines a source group called *source* which owns a prototype and includes the node [8,100]. The coordinates [8,100] correspond to the last two numbers of the IP address used by OMF to access the node (respectively 192.168.**8.100**). In order to separate this management interface from the interface dedicated to the experiment, a different IP address on each node has to be assigned (respectively 192.168.**0.100** for wlan0).

A prototype is a kind of application which is used to generate traffic flow. This prototype is defined as well as all its parameters. The source group is set into 'master' mode (acting as an access point). The element *packetSize* is expressed in bytes, whereas the element *rate* is in bits per second (bps).

```
#
# B) Define the 'receiver' group, which has the unique
#    node [8,102]. Nodes in this group will execute the
#    application 'test:proto:receiver'
#
defGroup('sink', [8,102]) {|node|
    node.prototype("test:proto:udp_receiver" , {
        'localHost' => '192.168.0.102'
    })
    node.net.w0.mode = "managed"
}
```

This second part of the experiment defines a *sink* group which is the source group counterpart. It therefore configures the node as 'managed' (client, also known as station). The prototype used is sink-specific and has for sole parameter, the localhost IP address of the node [8,102].

```
#
# C)  Configure the wireless interfaces of All the Nodes
#     involved in this experiment.
#
allGroups.net.w0 { |w|
   w.type = 'g'
   w.channel = "6"
   w.essid = "helloworld"
   w.ip = "%192.168.0.%y" # the '%' triggers some substitutions
}
```

This third slice declares the common parameters of the wireless interfaces of each node.

```
#
# D)  When all the nodes are turned On and all the
#          applications are installed and ready, we can
#          start to perform the experiment
#
whenAllInstalled() { |node|
    wait 30
    allGroups.startApplications
    wait 20
    allGroups.stopApplications
    wait 10
    Experiment.done
}
```

The last part is used to set up the nodes, to start and to stop the experiment. It is important to note that the statement *stopApplications* is strictly necessary to allow OMF to close the experiment and properly save the traces recorded from the traffics.

## 4.6 Why this version of OMF?

The University Federico II has installed the version 4.0 and will not consider installing the latest one (5.2). Indeed, the current version properly fits in with the needs and hardware that the university is currently owning. The university has two types of nodes and that is the reason why the university is currently not considering upgrading the installed version. The problem with the most recent version of OMF is the memory requirements to run the software (the node agent) on the nodes. This new OMF software now is consuming 25 MB instead of 15 MB with earlier release. The new version could be installed on the new nodes but not on the old ones! So, because of the lack of memory of the old nodes, the university prefers to stick to the currently deployed version. That basically means they cannot support all the new features and the new discovered bugs are not fixed unless if developers patch them by hand, which would require significant effort.

## 4.7 Database settings

When we first had to design our experiments, some problems appeared while no conclusive results showed up. After investigating our experiments again and again and after checking every single word of it, we found there was an external problem. Information about the hardware used in the testbed nodes was missing in the OMF database and performing an experiment without them has a wrong node

configuration as a consequence. Therefore, we added information in the database of OMF to fill the gap. The database is a Mysql database, version 5.0.51. The data access was granted with the user name *orbit*.

```
%> namur@omf-console:~$ mysql -u orbit -p
%> Enter password:
```

We then switch to the specific schema using

```
%> mysql> \u inventory
%> Database changed
```

All the tables are available from now on. The database is currently designed in ten tables. As shown in the next figure, the database structure is currently designed such as:

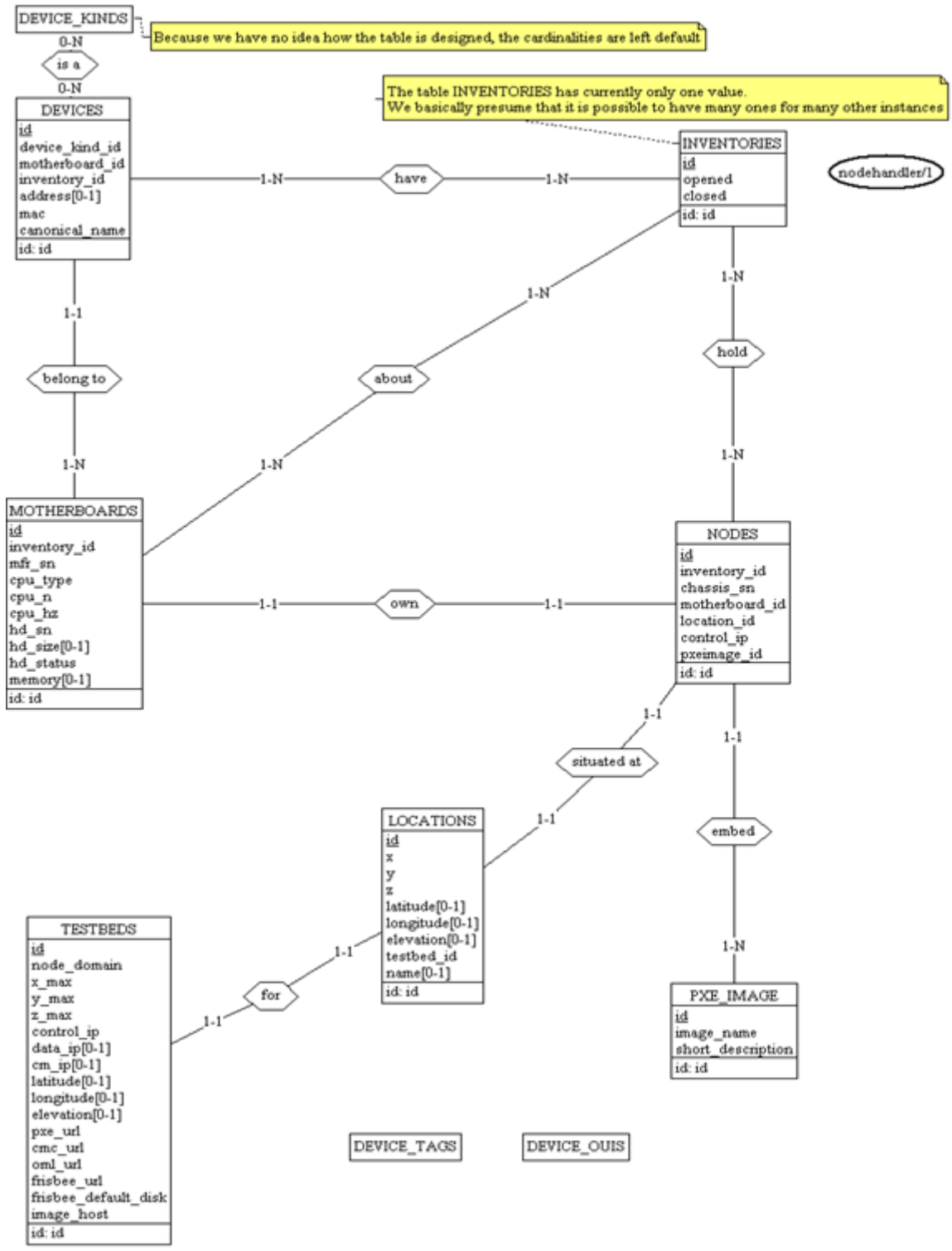



Figure 4.4: Structure of the OMF database

**INVENTORIES** The table lists the time information of the processes used in the environment.

**LOCATIONS** This table lists the position of the nodes (through the definition of coordinates) and the location of the testbed (thanks to the definition of the world location).

**MOTHERBOARDS** This table details the information related to the specification of the motherboards, hold by the nodes and associated to the network interfaces.

**NODES** The table specifies the characteristics of the nodes along with their configuration settings.

**PXEIMAGES** This tables records all the information for the disk images used to be loaded on the wireless device as an operating system.

**TESTBEDS** This table enumerates all the information over the testbed and its location.

**DEVICE_KINDS, DEVICE_OUIS & DEVICE_TAGS** Those tables store information that is useful only for the inventory. Those tables are optional [cO09c]. On the base of all those information, we had to add: the MAC addresses of the network interfaces (needed by *iptables* during the experiments) and the other basic information about the interface the IP-addresses of the nodes (Ethernet card).

## 4.8 Our contribution

Our first contribution to the OMF part was to fill the WILE-E database in with the nodes information. Since the topology depicted in the experiment description can not be applied without that information, which would result in a wrong node agent configuration, this phase was needed. Then we designed and performed new experiments to make sure the testbed was working fine. That enabled us to discover that the current version of OMF was too old to support the supposedly working features (implemented in any newer version). Finally, our last and biggest contribution was to make the XSL to automatically create the related experiment from the XML.

# Chapter 5

# The automatic generation

This chapter discusses the choice of the XML language to represent a network experiment scenario. It introduces the concepts and shows that the language is adapted to our needs. In the next point, the completeness of our approach is developed. The chapter also includes a list of various projects related to network experiment modelling with which we could compare and improve our tools. After this section, we present the solution we have built from an existing XML project. Finally, it depicts the difficulties we had to face and it explains the methodology we followed.

## 5.1 What is XML?

The XML [Con10] language is based on text-format tags and was created by Dan Connolly. The main use of XML is to transfer data on the Internet in a generic format between two hosts or servers. In a similar way than for HTML data is encapsulated into tags (see Figure 5.1). The language is said to be "extensible" because the user defines himself/herself his/her own tags and the entire layout s/he would like to give to his/her document. The two properties of an XML document are to be well-formed and validated. The XML has its own way to ensure it is indeed well-formed, but requires an outer agent to prove its validation.

This document has several ways to be validated. There is the old way of the Document Type Definition (DTD), allowing to define the different names and types that can be used in the XML. Or, there is a more recent version for the validation of XML documents called the XSD schema. The XSD schema has its definition closer to the XML one, which ensures the designer to keep the same set of mind for constructing his/her validators. The XSD schema is yet way more powerful than its counterpart DTD, not only because it can define the whole layout of the document and the type of elements in the

```
<root>
  <car>
    <owner>John Smith</owner>
    <year>2007</year>
    <fuel>diesel</fuel>
    <color>red</color>
  </car>
  <car>
    <owner>Alberto Rigota</owner>
    <year>2010</year>
    <fuel>petrol</fuel>
    <color>black</color>
  </car>
</root>
```

Figure 5.1: An example of a listing of cars

document, but also because it can make some reference-constraints in the whole document between those elements as well as it is possible to determine how many times a single element appears.

## 5.2 XML for this project

Finally, both the XML and the XSD schema are useful to define a set of data (XML) and to ensure those data are properly defined (valid) and well-formed. We decided to use an XML-based description language rather than a programming language for two reasons:

- The XML code is easy to understand, process, and extend.
- The descriptions in XML are more likely to be interchanged with other projects than scripts in a given special language.

## 5.3 XSL

XSL is a set of recommendations to realize some "transformations" on a XML document to produce another type of document (e.g., Hypertext Markup Language (HTML), XML, text). Indeed, the XSL is the sheet defining the style that will be applied to the XML.

This job is performed by a Extensible Stylesheet Language Transformations (XSLT) processor that takes a file which holds the data to translate (XML file) and a file with transformations rules (XSL file written in XML) and produces a third file in a specified format. In other words, from the data contained in the XML, the XSL is able to build a brand new file from the rules and laws defined in the style sheet.

The XSL sheets are mainly used on the Internet to produce HTML document that will be interpreted by a browser and then displayed.

Although the word "transformation" is not very relevant because the process does not change a file into another one (it creates a new file without any modification to the first one), this is this word that was accepted.

## 5.4 Completeness of our work

In this section we will discuss the completeness of our approach.

The completeness such as we define it, represents the capacity to consider all the common elements from the both platforms to be included in our work (see Figure 5.2). There is actually no formal nor semi-formal way to precisely prove that our work is currently complete and takes into account everything that should be taken care of.

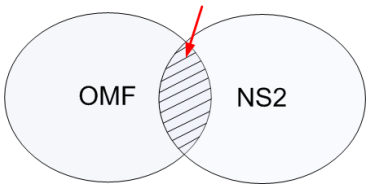



Figure 5.2: Common part of OMF and NS2 platforms

In order to approach the languages, we first had to delimit the parts we wanted to consider in both OMF and NS2. To do so, we considered the NS2 wireless functionalities as well as the OMF wireless feature. We started from the work done by Professor Canonico [RCG03], and we assumed his work was complete and gathered all the concepts to run a experiment (his work was made for a wired environment). His work provided us with the basics and a starting point to our work. Since we worked on the

wireless environment, many elements present in the wired setting are also available in the wireless one. We just had to add our wireless concepts for NS2 and OMF and remove the wired-specific components. Concerning the OMF platform, we also focused on the wireless features but in this case, we started from scratch.

We delimited the conceptual elements scope by taking the common part of the sets of concepts NS2 and OMF on which we applied the methodology described in Section 5.13.

The conceivable experiments available from our language are the ones defined in the language syntax constraints, i.e., XSD. If the experiment descriptions respect those restrictions, then our language and its translation processes can represent and convert them in a valid way for the concerned platforms.

We finished the process by the elaboration of a representative set of experiments that used most of the characteristics we implemented in our tool (see Section 5.8). The different steps were:

1. Imagine the experiments using the different representative features
2. Create the well-formed (XML standard) and valid (respecting the XSD specification) XML files
3. Automatically generate the experiment descriptions runnable on the different platform, thanks to our tool
4. Run those experiments and finally check the results compared to the expected ones

## 5.5 Related works

### 5.5.1 XML Description Language for Web−based Network Simulation

The work [RCG03], created by Professor Roberto Canonico from the University Federico II of Naples, has the ambition to model a network from a generic description. This work is composed of four distinguishable elements:

- a description of the topology
- a description of the different traffic flows in the experiment
- a description of the events
- a description of the different files created

The files created in his work XML Description Language for Web−based Network Simulation (XDLWNS) are divided in three parts:

- The network description
- The traffic description
- The simulation commands

Each of those parts implements one or two of the logic parts defined above. This work was the first step of ours, since we made some modifications in the schema (XSD) provided to us.

### 5.5.2 LETSQoS scenario generator

The project LETSQoS scenario generator [OHH02], a joint venture from the Darmstadt University of Technology and the University of Munich was created to model the same type of network (still without wireless) but using other languages than XML, which is called Graphical Modeling Framework (GMF) [Ecl10]. It is a graphic model of the network and tends to generate the experiments after designing the topology and the setup of the parameters. But the creators claimed that if they reach the limits of their own language, they will then switch to XML.

### 5.5.3 Modeling Computer Networks for Emulation

This scientific report [DHR] gives some techniques to describe an emulation scenario defining the network topology, link parameters and different ways to represent dynamic changes. The authors propose as well a description language based on XML as possible solution to a future automatic generation of description file for an emulator.

We compared this work to ours taking into account the differences we observed to improve our project.

### 5.5.4 Another Modeling language

Another Modeling Language (ANML) [Kid02] is a language created for describing communication network models. A model can simply be the representation of a network topology with specific properties. For example, a model could be three networks defining some nodes linked together inside each network and those networks are interconnected by some links with specific bandwidth. These models have to be compatible with some ANML schemas, like the DTD for XML, defined by the application developers or by the users themselves.

ANML schema language is used to define the structure: it specifies which components and attributes can exist in the model. These schemas describe the links, components that can be hold in the model, the associated attributes (properties) having to be used in the model and the composition rules of what we are allowed to model.

Furthermore, ANML language is based on Data Manipulation Language (DML), which is the base of Graphic Modeling Language (GML), used in LETSQoS scenario generator [OHH02]) and in XML. The syntax of ANML comes from DML, while the logical structure comes from XML. The ANML language

is formally defined thanks to an Extended Backus-Naur Form (EBNF) grammar [Gar10] represented by a set of rules called productions. Regarding the schema language, it draws of the DTD. Finally, the way they considered the representation of a topology could be interesting and is adapted for large scale networks. However, by comparison to our approach, it seems to be a bit more complicated but they have the advantages that they can reuse several times the same component defined only once (e.g., a router with a proc_delay = 0.000005, while in our work we have to copy/paste the same code). Moreover the dynamic part is not considered in their project.

As we were searching for an easy and complete way to model an experiment, we have chosen to take the XML language which allowed us to define what we wanted.

### 5.5.5 Conclusion

There may have been many other ways of modeling a network in a programming language, a scripting language or a graphical language, but XML by its own is probably the best one to model and describe a design as general as possible. It is that quality we want to stress, to motivate the use of XML in our own project.

## 5.6 Analysis and comparison with other projects

In analysing other works and projects, we got some comparison elements with XDLWNS project, which enabled us to add or transforms some features in it. In this way, we selected the best ideas to include in our work (based od XDLWNS).

### 5.6.1 LETSQoS scenario generator

The LETSQoS scenario generator project [OHH02] aims to develop a mean to automatically generate scenarios for simulation (e.g., with NS2) or for real-world experiments on a testbed which represents the same goal as ours. However, the major differences reside in the facts that:

- they do not manage wireless topologies
- they do not manage events and trigger actions
- they are using a language quite different from XML
- topology design is made by using a graphical interface (GUI)
  Due to the available time in our project, this goal has a low priority. We are considering that user will have to fill in the XML file manually.

#### General topology description

The authors of the LETSQoS scenario generator have described in details the syntax used in their topology file. Thanks to this information, we had the possibility to compare the common concepts between LETQoS and XDLWNS projects, that allows us to improve our modeling and offers the possibility to evaluate both approaches. The enumerated elements below (Table 5.1) have been considered on their presence irrespective of their implementation form.

In LETSQoS scenario generator, the authors chose to use GML without being sure that was the best way. Maybe using XML as a file format would have been a better choice. If GML parsing or validation becomes a bottleneck, then they might switch to XML but they did not.

So, it confirms that our XML-based approach seems to be a good idea. Nevertheless, this language, being more generic and verbose, needs a more complex translation process because we use a lot of tags and attributes to express the same information than in GML, a more specific one, which is suitable for modeling network graphs in a very straightforward way.

**Conclusion**: The same way is considered to represent the network in both projects. We have noticed some slight differences that can be changed easily.

| LETSQoS scenario generator | Table reference |
|---|---|
| Node | 5.2 |
| Link | 5.3 |
| CBR flow | 5.4 |
| Traffic | 5.5 |
| Packet | 5.6 |
| Comment | Implemented but not detailed in this document |
| Creator | Not implemented: Information added |

Table 5.1: Topology information

| LETSQoS scenario generator | XDLWNS project |
|---|---|
| ID | Implemented |
| Type | Implemented (e.g., Border Router or End System attributes) |
| Properties (e.g., RSVP capable, properties "Queuing = RED") | Partially implemented (e.g., RSVP capable not present) |
| Position (x,y) | Implemented |
| Label (additional) | Not implemented: Information added (e.g: examples of name: "source","relay",...). In our case, this information is mandatory |
| Traffic: Information about the traffic coming from this node. It is expressed by a simple string. (e.g., traffic "50% WWW, 50% IPTel") | Partially implemented but on a different level. Traffic at application layer cannot be expressed but can be formulated through a lower abstraction level (e.g., CBR flow generator, exponential,...) |

Table 5.2: Node information

| LETSQoS scenario generator | XDLWNS project |
|---|---|
| Direction (link unidirectional or bidirectional) | Not implemented: information added (e.g., full duplex or not) |
| From/To | Not implemented but the name "From/To" in LETSQoS scenario generator is ambiguous: in the case of unidirectional link, a node is both source and destination. It is not relevant in wireless environment to consider mono-directional link. If no link is specified in wireless, then the nodes in the transmission range of each other using the same channel can communicate together. The unique case we define links is to restrict the network communication to these sole links and to prevent any other data transmission. |
| Label (additional) | Not implemented and not useful |
| Properties (e.g., delay model,...) | Implemented to specify a static value for delay and bandwidth. In the case we keep information about wired link, we should consider the case using a dynamic delay model. |
| Bandwidth | Implemented in the properties |

Table 5.3: Link information

| LETSQoS scenario generator | XDLWNS project |
|---|---|
| Start Time | Implemented |
| End Time | Implemented |
| Interval between two Packets | Implemented |
| Target node, target port | Implemented |

Table 5.4: CBR flow

| LETSQoS scenario generator | XDLWNS project |
|---|---|
| Period Used with 'Burst Time', and 'idle Time' | Number of Bytes sent Not available. We consider rather a rate property |
| Source Node, port | Implemented |
| Target Node, port | Implemented |

Table 5.5: Traffic intensity layer

**Traffic layer**

In this part, we analyze traffic description by taking into account different levels of abstraction to express IP traffic. Those levels are characterized by the layers presented below. This comparison enables us to check if our approach based on the work XDLWNS is flexible enough to express different types of traffic.

**Application Mix Layer** : The mix of the different applications running on one node is described. An edge node could, e.g., be characterized as giving access to 100 users generating web traffic, 50 File Transfert protocol (FTP) users and 150 voice-over-IP sessions. As mentioned above, it is not implemented at this abstraction level. However, it might be a good idea to express traffic this way.

**Session Layer** : A complete session consisting of possibly more than one flow is described. A web session, for example, is characterized by user parameters and the characteristics of the reading times (times between the download of two web pages). It is never used.

**Flow Layer** : The flow layer, a higher and far more abstract layer than the packet and intensity layers, describes individual flows.

- Flow Type User Data Protocol (UDP) / Transmission Control Protocol (TCP)
- Flow Parameters

Table 5.4 lists parameters describing a Constant Bit Rate (CBR) flow.

**Traffic Intensity Layer** : The traffic intensity layer (Table 5.5) can be seen as an aggregation of several packets that fall in one period (e.g., one second). The time is split into equal length periods and traffic is characterized by the amount of data sent in these periods.

**Packet Layer** :Traffic on the packet layer (Table 5.6) is described by specifying the properties of individual packets.

| LETSQoS scenario generator | XDLWNS project |
|---|---|
| Packet Generation Time | Not available |
| Packet Size | Implemented |
| Source Node, source port | Implemented |
| Target Node, source port | Implemented |

Table 5.6: Packet layer

**Traffic models**

A traffic model describes how the traffic of one type is generated. Most of the relevant traffic models of the project are synthetic models: they use probability distributions like exponential, Poisson, Pareto to generate traffic on one (or more) traffic layer(s).

In NS2 and in OMF, synthetic traffic model like exponential, expo_on-off, can be used, while the Pareto and the Poisson models can not be represented because of their availability on only one platform.

**Conclusion**

Although the application mix layer and session layer are never used in our approach, the traffic for three others layers are used in XDLWNS with some differences. Depending on the research needs, the implementation of the application mix layer would be a good idea. Both LETSQoS and XDLWNS do not present any wireless concept.

### 5.6.2 Modeling Computer Networks for Emulation

A scientific report entitled *Modeling Computer Networks for Emulation* written by Daniel Herrscher [DHR] has as main objective to provide a way to express a scenario in an appropriate description language to generate automatically an experiment. The authors analyzed the requirements for an emulation scenario description language and proposed a possible solution.

Our work looks for the best way to express network experiments for testbeds and simulators in a generic language. The work accomplished by Professor R. Canonico in his paper [RCG03] concerns only simulators while D. Herrscher deals with emulators, which are a bit different (Refer to Chapter 2). Although those works have been done to be used by different tools, their goals are quite similar (i.e. generic description of experirments). That is why it is relevant to compare them and to keep the best ideas of both. Since the information required by a simulator is different from the one used by an emulator (see Section 2.5), we made a comparison based on this knowledge.

> "The scenario specification has to describe both the network topology and certain link properties. The topology description concept has to include point-to-point connections as well as multipoint-connections like LANs. Regarding the connections, all properties affecting the network performance in any way have to be modeled". [DHR] – Introduction

This is what both Mr. Canonico and Mr. Herrscher represented. They indeed made the same kind of research on links between nodes.

These two works do not treat the wireless component of a network and we have added this functionality in our tool.

> "Since in real networks, some properties will change over time, a realistic network model has to reflect such changes as well. We propose a modeling language that includes these parameters and offers concepts for dynamic changes." [DHR] – Introduction

This is one of the most interesting parts of the document. Indeed, XDLWNS made six years ago did not manage the changing properties over time and therefore lacks this important part. In the following part, we analyzed the tips the paper proposes to model and checked how we could implement those ideas in the continuation of XDLWNS project.

**Parameters to include in a network description** Here is a set of recommended parameters for modeling.

> "Bandwidth Limitation, Delay (propagation delay, serialization delay, queuing delay), packet loss must be included in a network description." [DHR] – Parameters Summary

For a point-to-point or multipoint link, the author recommends the following set of parameters to be modeled:

- Bandwidth limitation
  - maximum bandwidth (bit/s)
  - queue length (bytes or packets)
  - type of bandwidth sharing (simplex, half-duplex, or duplex)
- Delay
  - fixed delay (ms)
  - variable delay (stochastic function leading to a millisecond value)
- Packet loss
  - probability value
  - additional burst loss model

XDLWNS takes into account those ones: the queue type that specifies the length, the technique and the RED configuration, the maximum bandwidth, the fixed delay and the variable delay used in traffic pattern. However, the packet loss is not considered but we could add this information either on a link or in a traffic generator.

**The dynamic parameters** Everything concerning the variable delay components always has to be modeled in a dynamic model. More generally, all the variable parameters are supposed to be part of the dynamic section. Some of those parameters could be:

- the packet loss (especially in wireless networks).
- the bandwidth limitation (may also vary, considering the adaptive bandwidth in 802.11b).
- the propagation delay (can change if a communication partner is moving).

It is important to note that events are not taken into account in the XDLWNS project. So we had to find how to express them. Both Mr. Herrscher and Mr. Canonico decided to use the XML-language as a network topology generic descriptor.

### Possibilities to model traffic and dynamic behavior

We analysed how the traffic can be represented in an adequate way. Here are several models using two different approaches:

**Time–based** At certain time, an update of a property is done.

**Packet–based** It describes the behavior of each packet (e.g., medium access).

**Time–based: the table model** the table consists of tuples with triggers and actions. An action is formed to update some parameters or some dynamic models. We used often *time values* or *packet counts* as triggered conditions.

This kind of representation can be useful to represent an update of bit rate of a wireless link if there are too many errors, or simply the start/stop application action at a given time (see [DHR] – Dynamic Parameters Table).

```
<bandwidth type = "half-duplex">
  <table base="ms">
    <action at="0"> <value>11000</value> </action>
    <action at="2000"> <value>5500</value> </action>
    <action at="4000"> <value>2000</value> </action>
    <action at="8000"> <value>1000</value> </action>
```

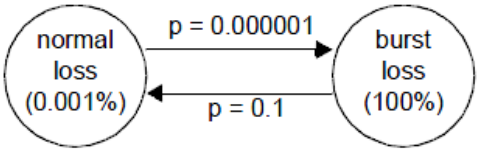



Figure 5.3: Markow state models gives the probability to switch of state

```
    <action at="10000"> <value>0</value> </action>
  </table>
</bandwidth>
```

Legend:

**Attribute *type*** of *bandwidth* tag takes its value in the set of simplex, half-duplex, full-duplex, that represents the kind of the link.

**Attribute *base*** of *table* tag takes its value in the set of ms, s. It is the unit of time used by the action tag.

**Attribute *at*** in the *action* tag is the time trigger to change the property of something.

This table is like a time-line: after 4 seconds, the property will vary from *5500* to *2000*.

**Packet–based: Gaussian distribution** it is appropriate to represent stochastic functions that explain how a value of a variable is updated by giving the properties of the variation (an expected value and the mean deviation).

It is useful to model the variation of the bandwidth of a wireless node moving away from an access point as it is a simulated environment (see [DHR] – Dynamic Parameters Gaussian Distribution). Conversely, we do not need to do the same for a wireless testbed because this variation is intrinsic to reality.

```
<variable_delay>
  <gaussian base="packets" mean="5" deviation="1" />
</variable_delay>
```

**Packet–based: Markov state models** this model, more powerful than the previous one, gives the probability to switch from a state into another one. It is particularly relevant to express something like a loss model or change of bandwidth over time.

We can imagine using this representation in combination of a table model in which each value is replaced by a Markov state model (at a certain time, a different state model will be used).

Figure 5.3 and the following example (see [DHR] – Dynamic Parameters Markov State Model) show a way to express a Markov state model about the burst loss behavior. Of course, other models could be considered for a packet-based representation.

```
<loss>
  <markov initial="normal" base="packets">
    <state id="normal">
      <value>0.00001</value>
      <transition id="burst" probability="0.000001"/>
    </state>
    <state id="burst">
      <value>1</value>
      <transition id="normal" probability="0.1" />
    </state>
  </markov>
</loss>
```

Our idea is to create a dynamic part, which would be a dedicated section in an experiment description. This section includes the actions that happen at different scheduled times (i.e., the time when the first log is recorded, the time to shut down a specific network interface, etc.) according to the will of the experimenter and the other actions that cannot be foreseen (i.e., the transmission rate which is adjusted based on error rate during the experiment).

This section could also be called the events subdivision because it determines the moment when an action should occur and it gathers all the special events that can take place in the experiment.

## 5.7 Our XSD solution

This section presents our solution to implement the wireless functionality. The documents in annex .1 will detail the elements we added, removed or modified from XDLWNS.

All the following elements in the next point are common to NS2 and OMF.

### 5.7.1 Result of XSD modifications

Because a schema is more readable and comprehensible than a source code, we details our project thanks to a graphical representation. We focus on the most important XML tags we used by explaining the semantic of them.

This graphical representation comes from the XML editor *Altova XMLSpy* [Alt10] which is a very good tool to edit and develop XML, XSD and XSLT. In the following figures, we adopted this representation:

Figure 5.4: Sequence of element(s)

Figure 5.5: Choice among the children


Figure 5.6: Add comments/notes to each element


Figure 5.7: Adding multiplicities on different kinds of element

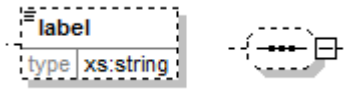



Figure 5.8: Different kind of optional element (multiplicity = 0..1)

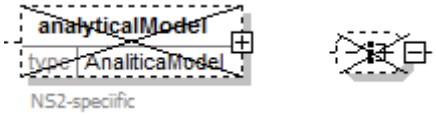



Figure 5.9: Different kinds of element that cannot be instantiated

The description of an experiment is organized in three main parts:

**networkDescription** holds the network topology description

**traffic** describes the configuration of the flows that will happen during the experiment

**simulationCommand** contains all the events and planned actions during the conduct of the experiment

Attribute *xmlName* has to contain the name of the XML file. This information is needed because the XSLT processor cannot access other data than those in the XML tree.

Figure 5.11 details the network Description part.
Element *as* is the Autonomous System that will contain some nodes. If the user is not familiar with this concept, s/he can put its *id* attribute to 1 and give a name that would represent a set of nodes. Element *topologyModel* holds all the information about the description on the wireless nodes, as shown in the next picture.

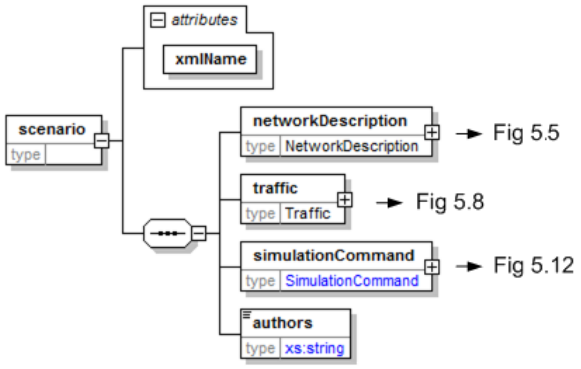



Figure 5.10: Scenario element and its children

Figure 5.12 concerns the information about a node, which represents a device in the real world.
Attribute *id* : unique number used as an identifier in the whole experiment
Attribute *name* : unique name in the whole experiment
Attributes *x,y* : the identifier of the node in the testbed and in the network simulator
Attributes *posX,posY* : the node absolute position in the infrastructure
Attribute *nomadicity*: the value of the random motion of the related node
Element *cfgNetInterface* : contains information about the configuration of the network interfaces
Attribute *mode* : values are "ad-hoc", "master" or "managed"
Attribute *type* : values are "a", "b" or "g" ("n" not yet available on the platforms)
Attribute *channel* : value of the radio channel (e.g., 1 to 11 for 802.11g - 802.11 only available though)
Attribute *essid* : name of the wireless connection
Attribute *device* : represents the network interface (w0 or w1)

Figure 5.13 shows the routing part of the network experiment.
Element *routingDaemon* : information about the routing protocol/static routes used by a specific node.
Element *aodv* : by specifying a network interface, AODV daemon will be executed on the node.
Figure 5.14 explains the protocol used on the node. We kept the layer 4 information which is exploited to configure the protocol type in both platforms.

- If a node is a traffic generator (source), then it has to specify the transport protocol used (UDP / TCP). For a node, *id* attribute of UDP or TCP elements is a unique number (inside the *protocol* element). However, in the same experiment two different nodes could have the same identifier.
- If a node is a receiver (sink), then it has to use the *Null* element
- If a node is only a relay, then let the *protocol* element empty

Let us take an example of an experiment composed by three nodes in which the first one (1) is both a source (UDP) and a sink. The second one (2) is only a sink and the last one is set up as a source (UDP). In this case, the configuration will be:

Node (1) = <udp id="1"> <Null id="2">
Node (2) = <Null id="1">
Node (3) = <udp id="1">

Figure 5.15 describes the traffic flow produced on each source node.
Element *pattern* : represents a flow between a source and a destination.
Attribute *id* : identifies a particular pattern in the whole experiment, that is why this number has to be unique.
Element *src*, *dst* : hold information to identify nodes. In case of need, it will be possible to extend the

*destination* element to take in account a broadcast sender.

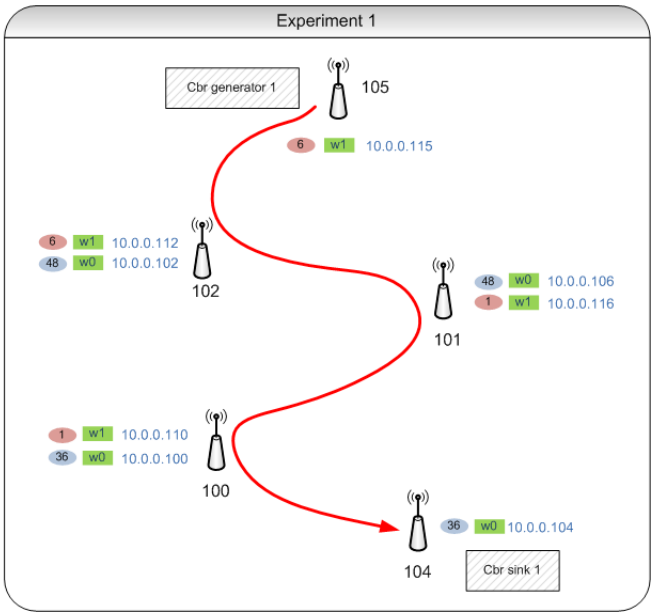
Figure 5.16 focus on the traffic pattern describing the flows.
Elements *src* and *dst* : contain enough information to retrieve the IP address of a network interface of a node.
Attribute *nodeId* : number which has to be the unique identifier of a node.
Attribute *netInterface* : the network interface (w0,w1,...). Currently WILE-E nodes are equipped with only two interfaces.
Element *trafficModel* : the child of the element *pattern* and is enough generic to be implemented in different ways in NS2 and in OMF.
At this moment, only *CBR* element (in UDP) is implemented in OMF and in NS2.

Figures 5.17 and 5.18 explain respectively the parameters required for a CBR flow generator and for a Expo_On_Off generator.
Attribute *packetSizeUnit* : has to take a value in the set "bits, Kbit, Mbit"
Attribute *rateUnit* : has to take a value in the set "bps, Kbps, Mbps"
Attribute *intervalUnit* : the value between two CBR flows

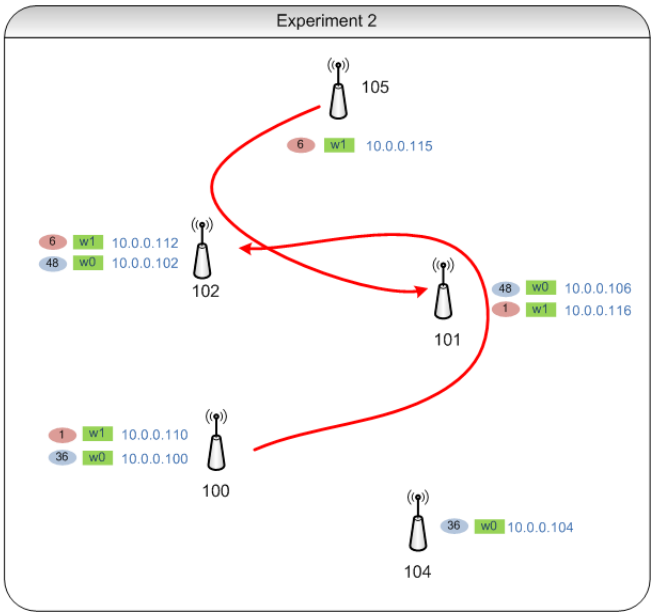
Figure 5.19 illustrates the commands to control the simulation.
Element *simulationCommand* : contains all the time-related components.
Attributes *startTime* and *stopTime* : refer to the start time / stop time of an experiment. In OMF, the user has not to be worried about the period of time needed by the devices initialization because the *startTime* attribute is the moment from which all the devices are already set up and ready to start the real "experiment" (not the initialization).
Attributes *startLog* and *stopLog* : are specific to *NS*2 and refers to the time of beginning / ending the log process for the simulator.
Element *actions* : kind of table in which each line represents an action that is triggered on a specific time. At this moment we manage the start/stop traffic but due to bugs in OMF and non implementation in NS2, *downInterface* and *upInterface* cannot be used.
Attribute *idTraffic* : number pointing out a specific pattern. at attribute specifies the time to accomplish the action.
Attribute *nodeId* : unique number and play the role of an node identifier.
Attribute *netInterface* : refers to the network interface (w0 or w1)

## 5.8 Performed experiments

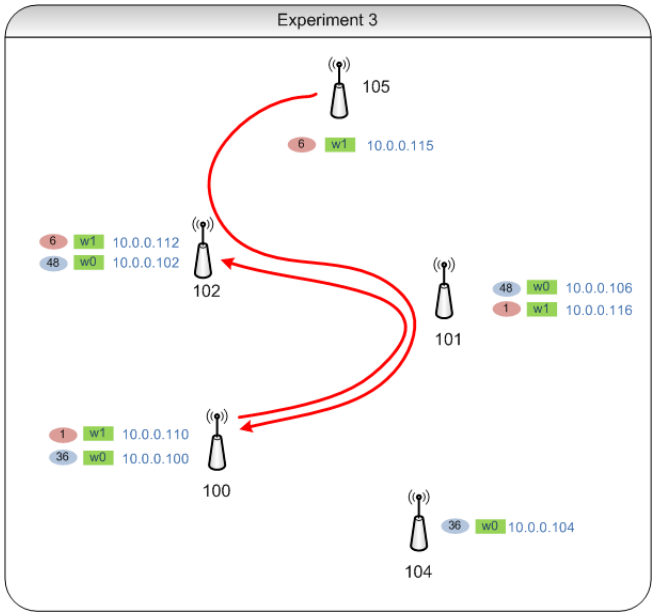
We finally performed five experiments to check the validity of our work. Those experiments have been selected because they are exploiting different OMF features like simple flow, multi-flows with multiple senders and multiple receivers. We think they positively represent the common set of features offered by the OMF platform and the NS2 simulator. Our tool makes it possible to express these kinds of experiments. Those experiments can be found in their XML format in the appendix, Section .5



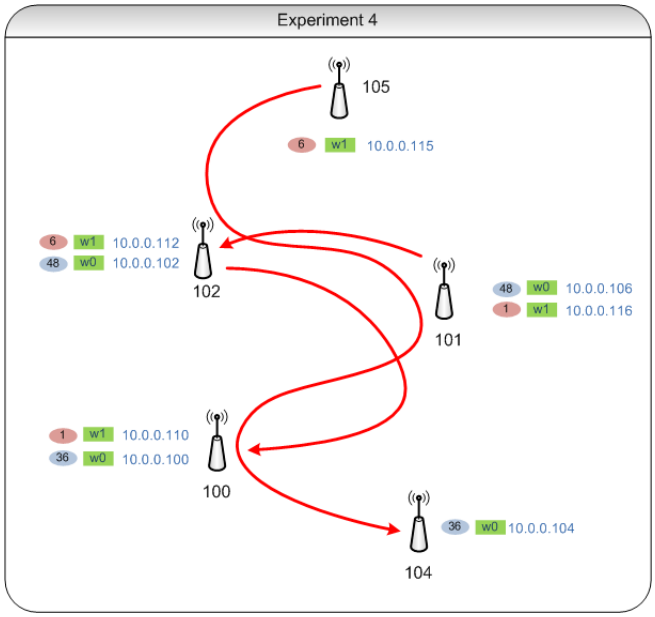

Experiment:
CBR UDP flow from 105 to 104
Result: successful (all the packets reached the destination)



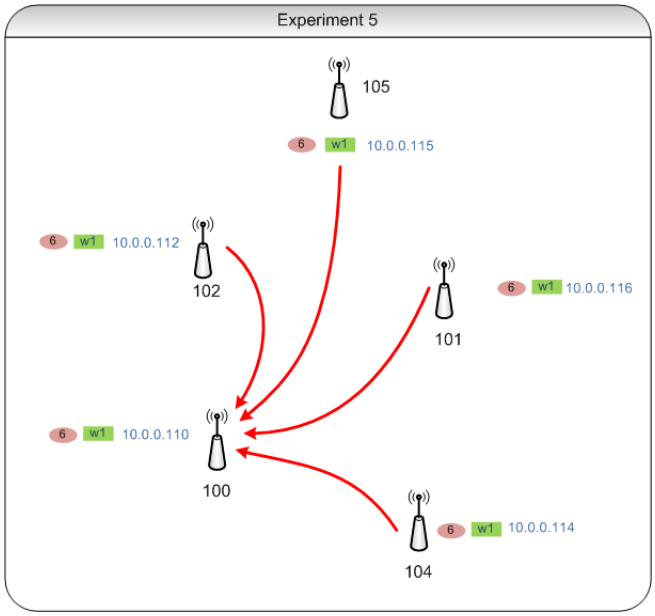

Experiment:
CBR UDP flow from 105 to 101
CBR UDP flow from 100 to 102
Result: successful (all the packets reached the destination)




Experiment:
CBR UDP flow from 105 to 100
CBR UDP flow from 100 to 102
Result: successful (all the packets reached the destination)




Experiment:
CBR UDP flow from 105 to 104
CBR UDP flow from 101 to 102
CBR UDP flow from 102 to 100
Result: successful (all the packets reached the destination)




Experiment:
CBR UDP flow from 105 to 100
CBR UDP flow from 101 to 100
CBR UDP flow from 102 to 100
CBR UDP flow from 104 to 100
Result: FAILED Only 3 flows reached the destination.
Some flows did not have been logged into the OMF database.

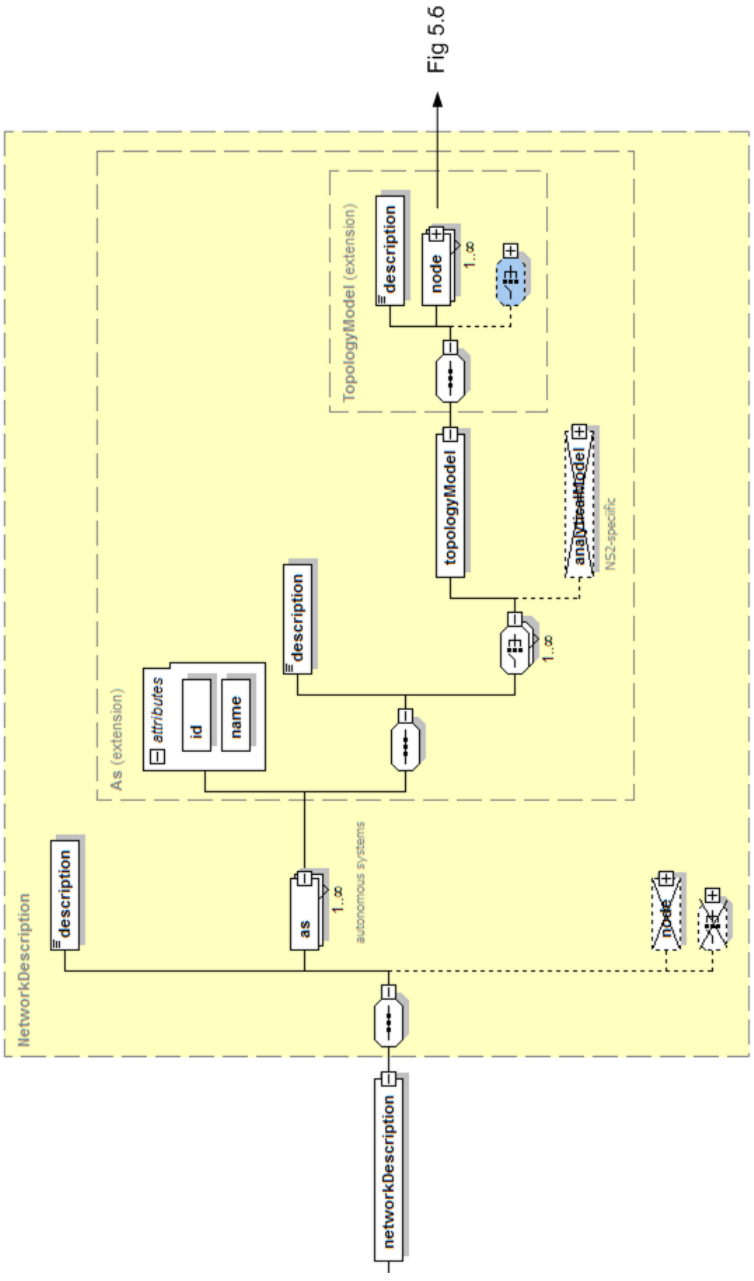



Figure 5.11: Network description element and its children

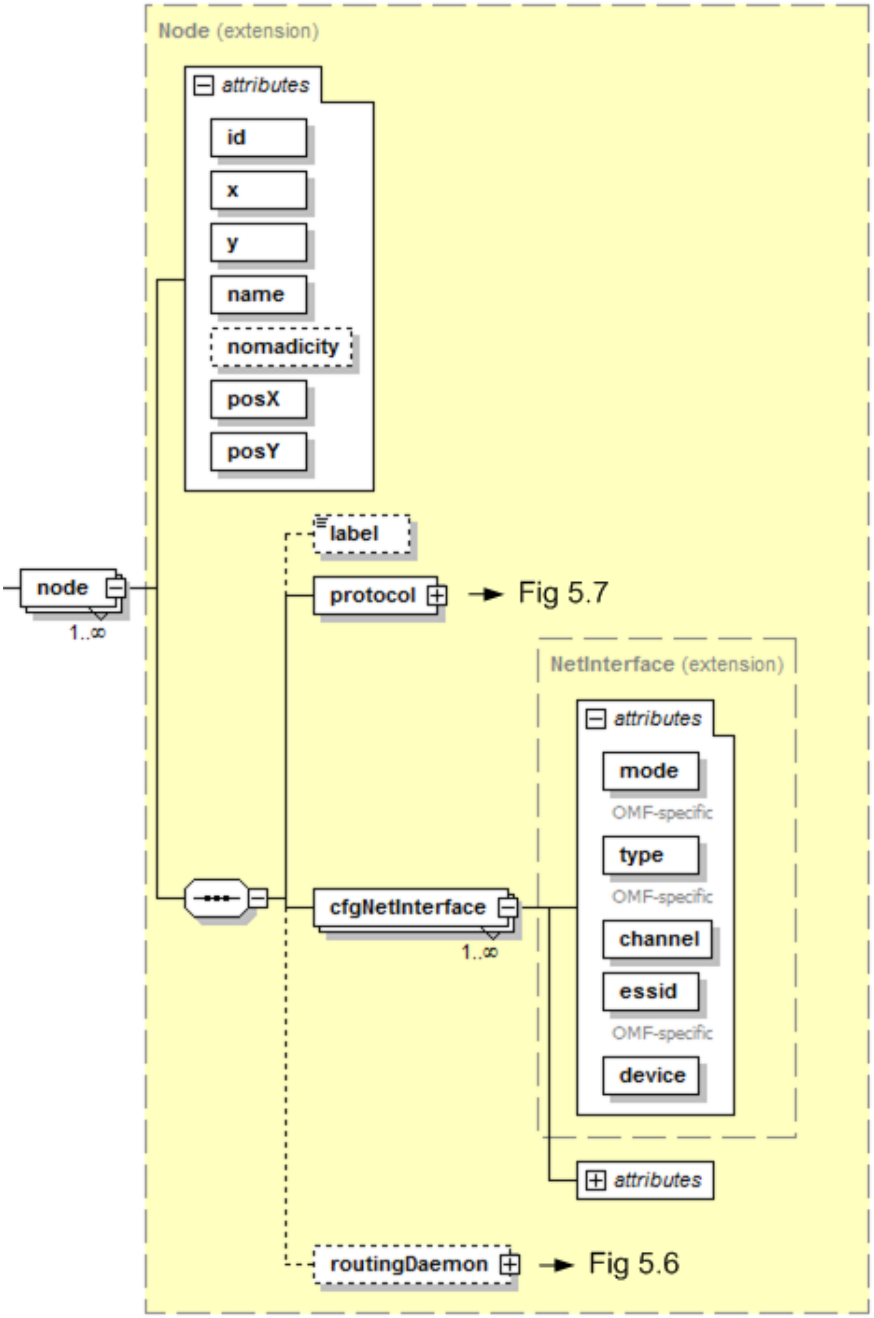



Figure 5.12: Node element and its children

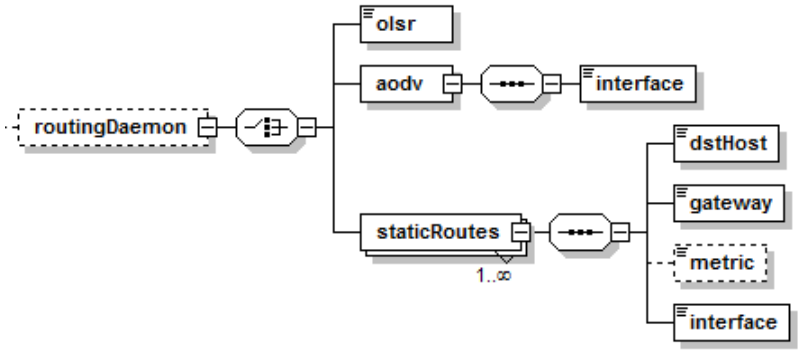



Figure 5.13: Routing element and its children

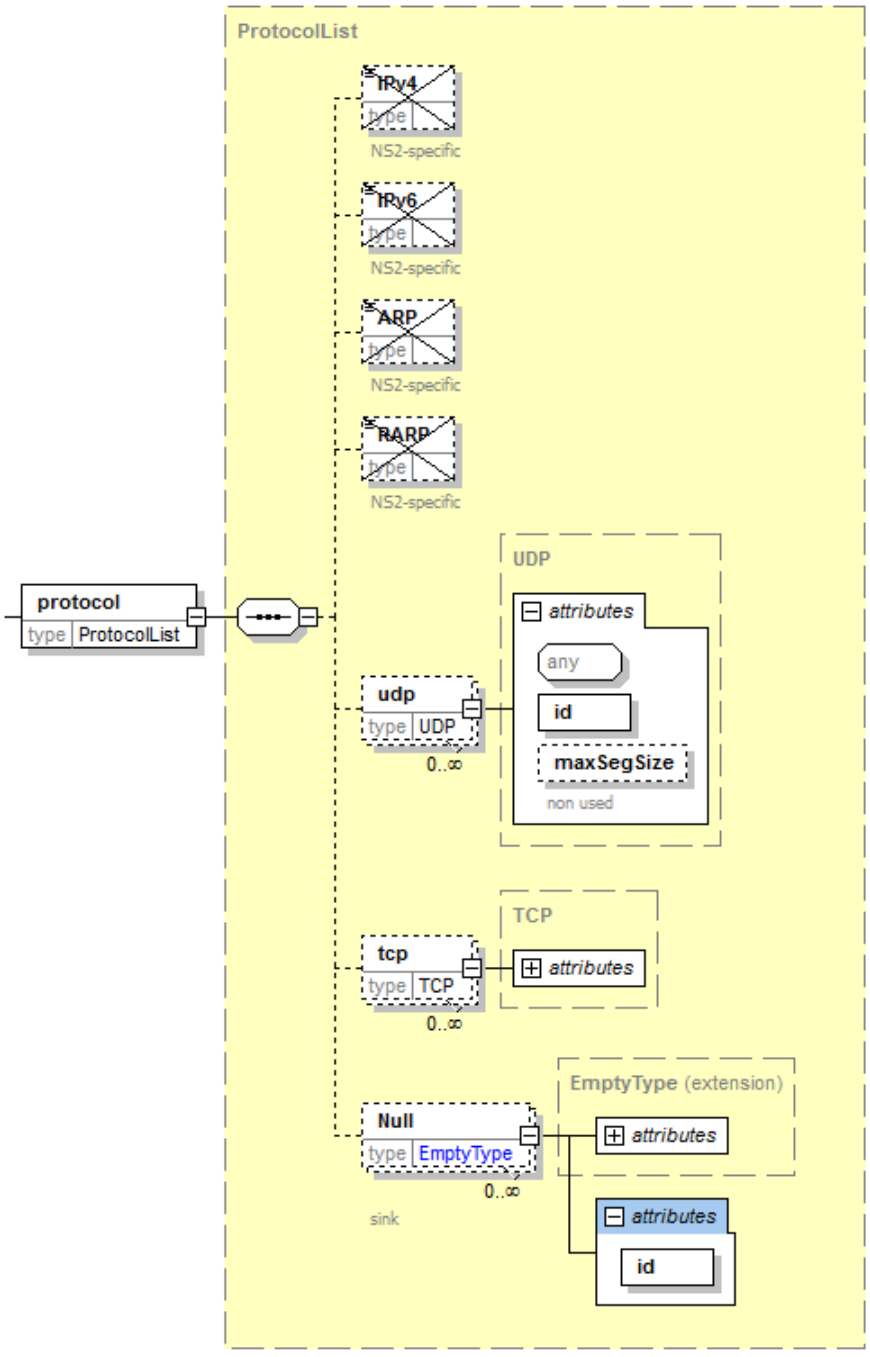



Figure 5.14: Protocol element and its children

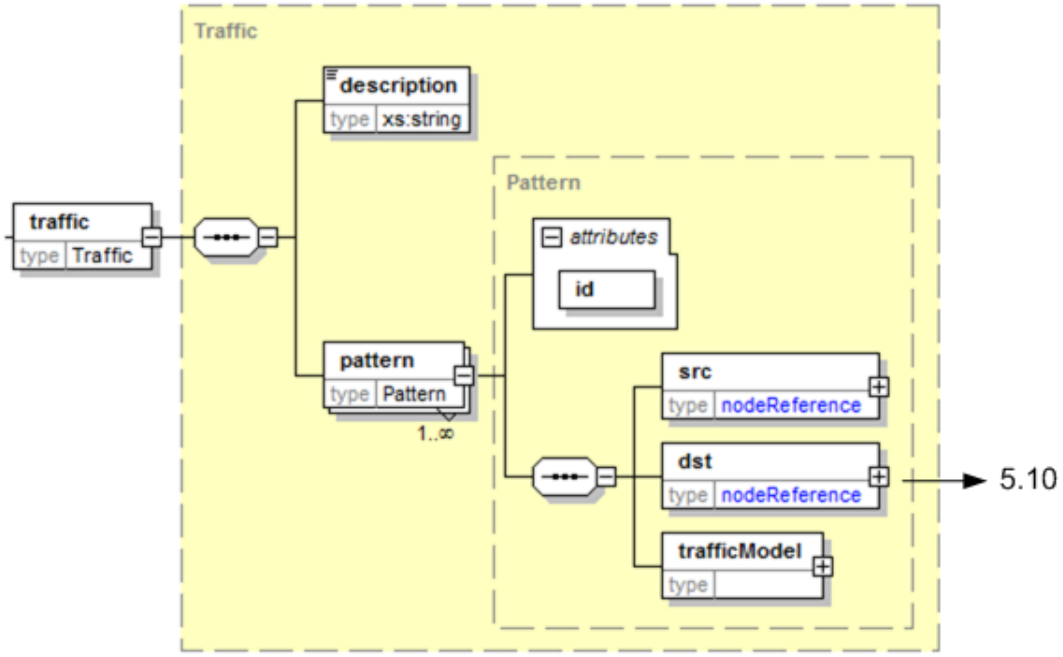



Figure 5.15: Traffic element and its children

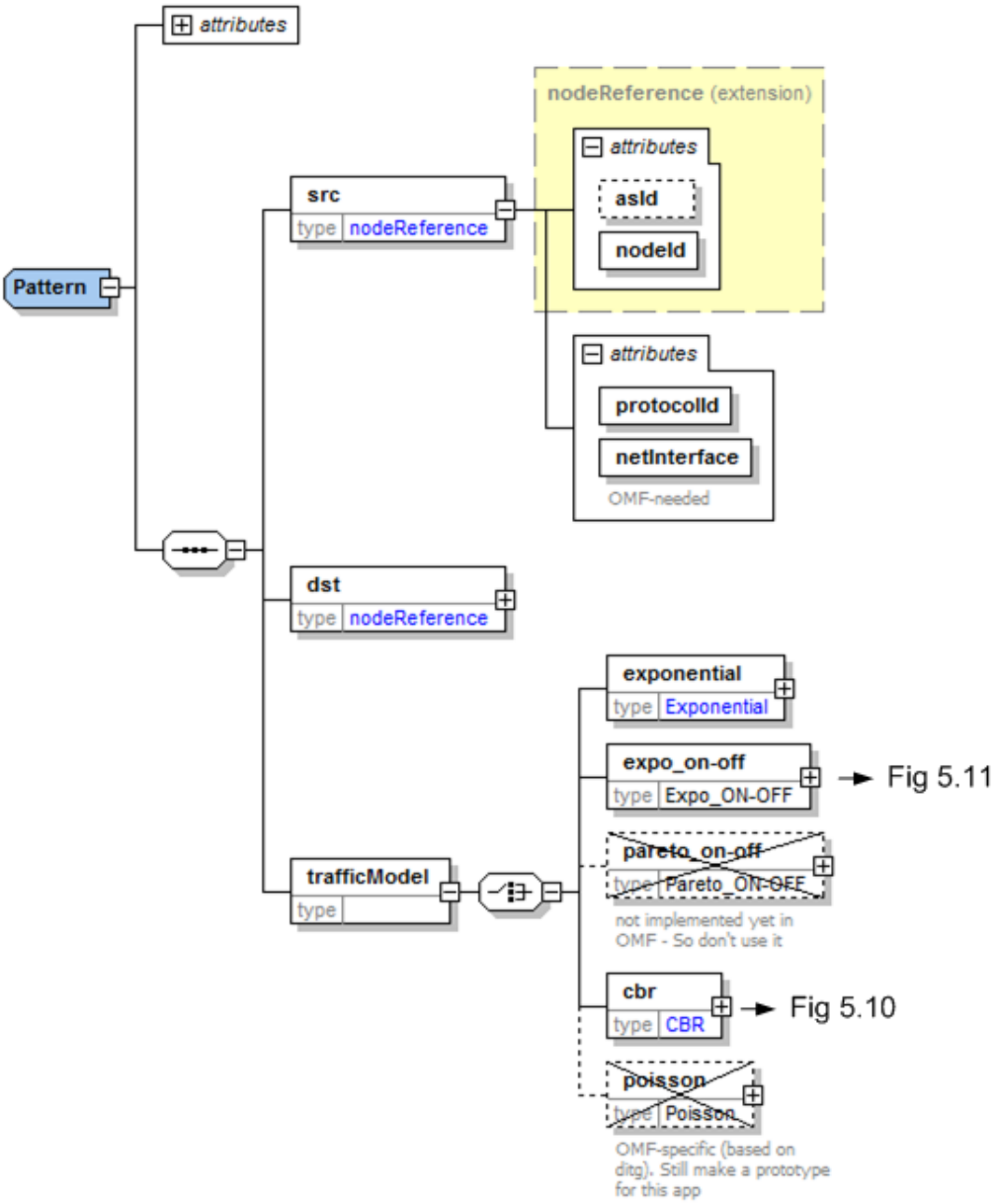



Figure 5.16: Pattern element and its children

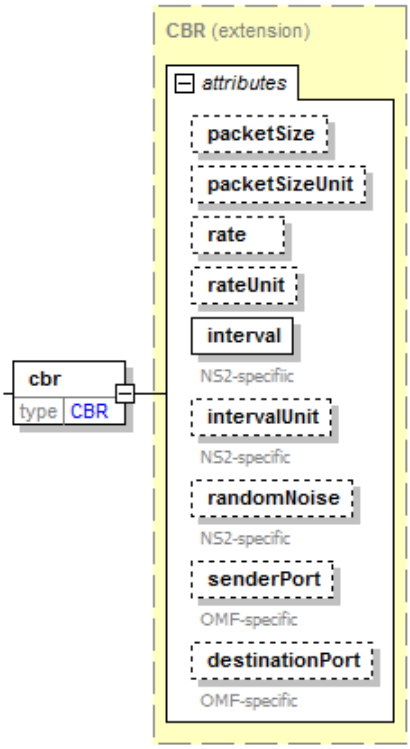



Figure 5.17: CBR element and its children

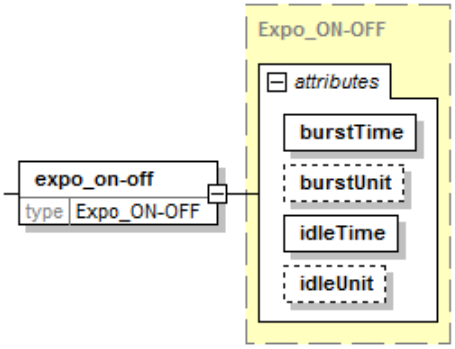



Figure 5.18: Expo_on_off element and its children

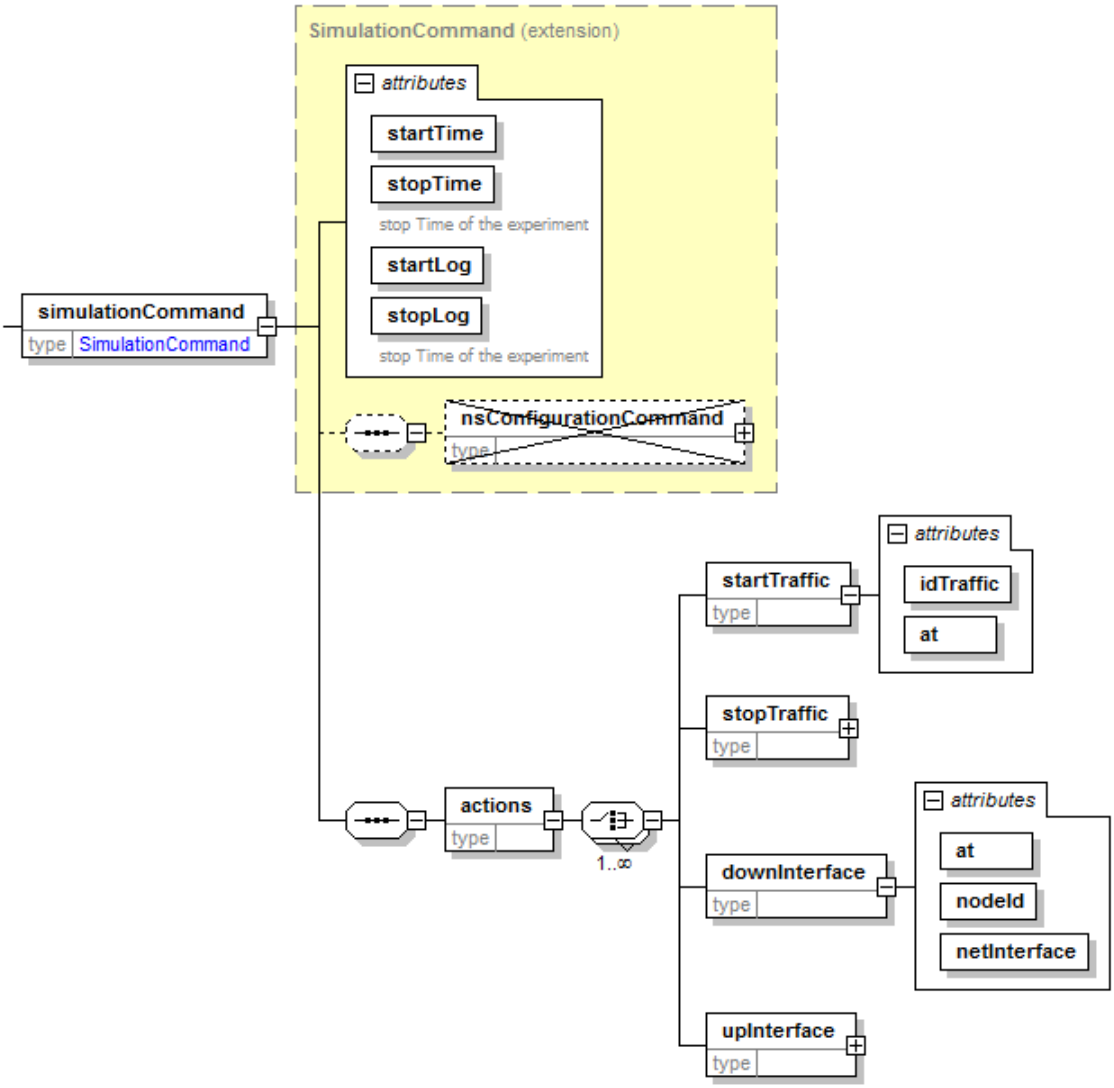



Figure 5.19: simulationCommand element and its children

## 5.9 OMF difficulties

We had to deal with an important problem during the whole project concerning OMF: the lack of documentation. The documentation provided on the website [cO09d] was incomplete and was sometimes not up to date. It therefore took a lot of time to master all the characteristics used in OMF. The best way to proceed, after reading the basics, consisted in reading a great number of existing experiments found on the website. Unfortunately, due to the particular structure used to present this information, which is based on the open source media Wiki, this task was not very obvious. Indeed, the information related to a given topic was scattered all over the website. So, the major drawback is that there is not a single place where the experiments are stored. Moreover, as a consequence of the point above, the wiki structure does not enable to organize the website like a tree (with a root and a hierarchy). It makes the navigation quite random and we never know what the next page will be like. Additionally, the provided examples were really straightforward and the attempt to make more complex experiments was a tricky task. It is difficult to find information as we do not know what it is possible to do or not.

We also faced a second problem of some importance, which is the source of most of the errors obtained. The OMF version running on WILE-E is not the most recent one and is considered to be deprecated from different points of view. As we will see, some errors are caused because some features are not yet implemented or due the fact that some bugs were discovered and subsequently corrected.

Here is a listing of problems or errors encountered during our experimentations with the OMF platform.

### 5.9.1 Output process failed

While trying to test some of our experiment scenarios, the result data were missing. The output process did not work properly and it may result in different situations:

- A socket bind error with no database file containing the experiment data (Space Quest 3 (SQ3) file, the OMF database file) generated.
- SQ3 database file is created, but there was no data in the receiver part ("in" table).
- SQ3 database file is created, but there was no data neither in the receiver part ("in" table), nor in the sender part ("out" table).

**Solution**: Once one gets incomplete results files or if those files are not created at all, we recommend to kill the service. It will start up again automatically. To do this, we had to run the following command line:

```
# sudo killall oml2-server
```

### 5.9.2 Infernal process loop

If the problem above did not happen, an other was causing pain using 100% of the processor. The process *oml2-server* went mad and the OMF platform was therefore unavailable.

**Solution**: There is no real solution to resolve the problem. So, to continue to use the testbed, we had to run the next command:

```
# sudo killall oml2-server
```

#### Oml2-server kill consequence

After killing the *oml2-server* process, we had to wait a few minutes before the next experiment execution. Here is the console output:

```
INFO init: NodeHandler Version 4.4.0 (1921)
INFO init: Experiment ID: wile-e_2009_12_16_09_20_24
FATAL service_call: Exception: ServiceException (http://192.168.8.200:5022/
      oml2/start?id=wilee&domain=wile-e)
INFO ExecApp: Application 'commServer' finished
INFO run: Experiment wile-e_2009_12_16_09_20_24 finished after 0:15
```

### 5.9.3 Version obsolescence problems

The following problems are due to the fact that the OMF installed version is obsolete, as the version is 4.4.0 whereas the latest version is 5.2.

#### Generator glitch

While trying to run some experiments, we discovered some recurrent errors. Those errors are either from the generator Distributed Internet Traffic Generator (DITG) or from the prototypes that configure it. Because of the lack of time at the end of the internship traineeship, we failed to determining what the exact answer to that problem is. As an alternative, we also used the second version of Orbit Traffic Generator (OTG), but some other errors arose. However, it seems that OTG2 is working for simple experiments. We contacted the OMF developers for an experiment that was supposed to run properly, and they told us our version of OMF was deprecated. So, we somehow believe that the generator has its part of responsibility in the generated errors as well as the prototypes have.

#### Interface shut-down error

We tried to turn off a wireless interface during an experiment with the command found in an experiment (mpath-experiment.rb [cMFO10]) on the OMF website. That is why we made some experiments using the following command

```
group('relay2').net.w0.down = 1
```

as found on the website. Each time, we got the same execution error:

```
ERROR EXECUTION_ERROR: Execution Error on node: 'n_8_101' - Error Message:
    CONFIGURE' with 'net/w0/down' Value to set parameter to
```

After checking the log files, it happened that *'CONFIGURE'* with *'net.w0.down'* is received properly on the Node Agent, but for some reasons it cannot be executed. This might be a bug that was fixed since then. We messaged the OMF developers, who have tested the same experiment in their testbed (OMF release 5.2) and it worked fine.

#### Add route to topology

Before creating experiments with dynamic routing protocols, in the beginning of our project, we wanted to add some static routes on each node to forward traffic to the destination. This could be done by using this experiment line [cO10c]:

```
net.w0.route({:op => 'add', :net => '10.1.0.5', :gw => '10.1.0.2', :mask =>
      '255.255.0.0'})
```

It happened that no error was generated but after experimenting some tests, we saw that no new route was added to the Linux routing table of each node. This might be a feature implemented in the latest OMF version. We tackled this problem by manually putting the following command in the OMF experiment description:

```
group('relay2_1').exec('route',['add','-host', '10.1.0.5', 'gw',
    '10.1.0.2','metric','2'])
```

#### Problem encountered with parameters modification during execution

While making some experiments in our early steps of understanding the OMF platform, we were asked to create an experiment which has to modify the bit rate of a wireless card during the experiment. We made that experiment using the few hints we read from the OMF website and tested what we had made up (the prototype used was "OTG2"). Although after several attempts without any reported problems and even if the experiment was correctly designed, the result did not come out. We messaged the OMF developers to know what was wrong with our experiment. It turned out that the experiment was properly designed but the version of OMF was too old to handle the problem. The current version is 4.4.0. The version that can deal with the new addings was the next one, which unfortunately cannot be set up due to the lack of nodes RAM memory. Because of the old version, some experiments were made impossible as the feature is not implemented in the current version installed so far. However, we alleviate the problem by correcting the version in adding the code from the trunks.

#### Node Application failure

One of the problems we had to deal with was the application run by the nodes crashing for no reason. The sole hypothesis we had to explain was the non up-to-date version of the platform installed.

This is the console output:

```
INFO exp: Request from Experiment Script: Wait for 2s....
INFO exp: Request from Experiment Script: Wait for 8s....
INFO exp: Request from Experiment Script: Wait for 30s....
ERROR NodeApp: ERROR: --udp:lo (unknown option)
ERROR NodeApp: Usage: otr2 [-h|--help [component]] [--udp:local_host=[name]]
ERROR NodeApp: [--udp:local_port=INT] [--udp:dst_host=[name]] [--udp:dst_port=INT]
ERROR NodeApp: [-p|--protocol udp] [-g|--sink udp|null] [--flow:id=INT]
ERROR NodeApp: [-d|--debug-level FIXED] [-l|--logfile FIXED] [-v|--version] [--exit]
ERROR NodeApp: [--pause] [--resume]
INFO exp: - Stop all the applications
INFO exp: Request from Experiment Script: Wait for 20s....
ERROR EXECUTION_ERROR: Execution Error on node: 'n_8_100' - Error Message: 'STDIN' with 'app:otr2 exit' -
     Error while writing to standard-IN of application 'app:otr2' (likely caused by a a call to 'sendMessage'
     or an update to a dynamic property)
INFO exp: ------ Now Stop the experiment
INFO Experiment: DONE!
Solution: reboot all the nodes
INFO exp: - Start all the applications
INFO exp: Request from Experiment Script: Wait for 45s....
ERROR NodeApp: ERROR: --cb (unknown option)
ERROR NodeApp: Usage: otg2 [-h|--help [component]] [--udp:local_host=[name]]
ERROR NodeApp: [--udp:local_port=INT] [--udp:dst_host=[name]] [--udp:dst_port=INT]
ERROR NodeApp: [--udp:broadcast=on|off] [--udp:nonblock=on|off] [--cbr:size=bytes]
ERROR NodeApp: [--cbr:interval=msec] [--cbr:rate=kbps] [-p|--protocol udp|null]
ERROR NodeApp: [-g|--generator cbr|expo] [--flow:id=INT] [-d|--debug-level FIXED]
ERROR NodeApp: [-l|--logfile FIXED] [-v|--version] [--exit] [--pause] [--resume]
ERROR NodeApp: Exception: Socket Bind Error
ERROR NodeApp: Exception: Socket Bind Error
ERROR NodeApp: ERROR: --udp:local_por (unknown option)
ERROR NodeApp: ERROR: --udp:loc (unknown option)
ERROR NodeApp: Usage: otr2 [-h|--help [component]] [--udp:local_host=[name]]
```

## 5.10 Remarks

This section compiled all miscellaneous kinds of useful information.

### 5.10.1 Node configuration

- **OLSR**
  To run OLSR [OLS10] on the recent nodes, since they are set up with only Read-Only partitions, a symbolic link that redirects a lock file into */tmp* directory has to be created by the following commands :

```
remountrw
localhost:/etc# ln -s /tmp/olsrd.conf.lock olsrd.conf.lock
remountro
```

- **Filtering Rules**
  On the oldest nodes : Remove the filtering rules (Executed automatically in the XSL files).

### 5.10.2 Test of the addLink feature

The functionality *addLink* is used in the definition of the topology in OMF. After defining which nodes are participating in the experiment we are creating, we then declare the links between those nodes. The *addLink* feature has the particularity to remove all the other links that are not explicitly defined between the first node of the *addLink* and all the others. Even though the *addLink* has that particularity, it is not properly detailed in the documentation made by the OMF developers on their website [cO10b].

```
Adds a link between nodes x and y and configures it with the
    characteristics defined in the 'spec'.

'spec' is a hash with the following valid keys {:rate , :per, :delay, :
    asymmetric}

:rate - set the link's rate in Mbps
:per - set the link's packet error rate in percent (0.3 for 30%)
:delay - set the link's delay in ms
:asymmetric - set if this link is asymmetric or not ('true' or 'false')

Note: Only 'asymmetric' has an effect on the link, a traffic shaping
    component is
       currently being added to OMF in order to implement the other
          attributes.
```

**Simple example of an addLink use:**

```
defTopology('mainTopology') { |topo|
    n1 -> n2
    myNodes = {
      "n1" => "[8,100]",
      "n2" => "[8,101]",
    }
    topo.addNodes(myNodes)
    topo.addLink("n1","n2")
}
```

This example declares two nodes, node *n1* and node *n2* and creates a link from the node *n1* to the node *n2*.

**Available features not implemented**

By using the current version (4.4.0) of OMF, some available features are still not implemented yet in the source code. However, no error is generated. It just does not produce absolutely any effect. This is the case for the link property provided in the *addLink* feature (describe above): 'per', 'rate', 'values' are not implemented yet.

### 5.10.3 Node location in NS2

To perform a common experiment with the same topology on OMF and NS2, the virtual static placement of the nodes in the simulator has to be similar to this of the real nodes. Indeed, their location is based on the configuration of the laboratory *'Informatica e Systemistica - Network lab'*. The reference point we choose, is on the top left corner when you enter into the laboratory. Then, in order to give a position to the other nodes, the squares on the roof (size: 50x50 cm) helped to determine the relative placement from the reference point to any other node.

The schema of the laboratory looks like the following plan:

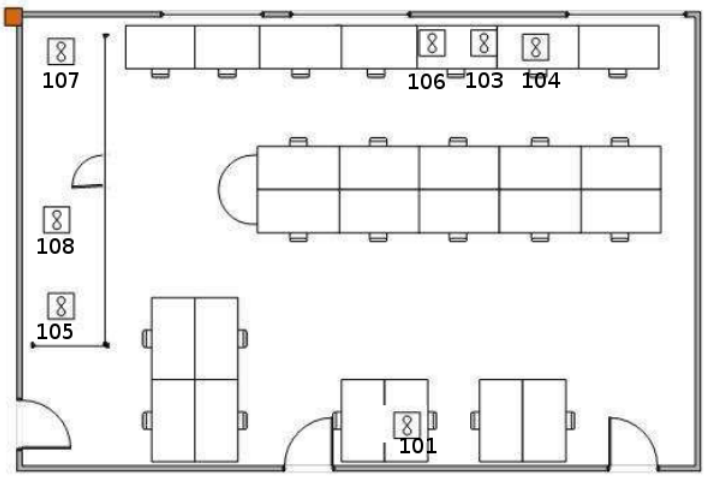



Figure 5.20: Structure of the network laboratory in Naples

## 5.11 XSLT processor

As previously explained, the transformation of XML documents into HTML, text or others formats is accomplished through the execution of an XSLT processor on this document. XSL files, which are written in the XML language, defined the rules to apply when the processor meets some specific tags in the source document.

```
<xsl:template match="networkDescription">
   <xsl:apply-templates select="description"/>
   <xsl:variable name="nodeNumber" select="count(//node)" />
## Number of nodes present in the experiment = <xsl:value-of select="$nodeNumber"/>
```

Figure 5.21: Example of a XSL part

Since classical XSLT processors implement only the World Wide Web Consortium (W3C) specifications, the choice in the set of available actions seems to be quite restrictive in specific cases. W3C being a great standard, it takes a while to add new concepts and functionalities in the XML standards.

In order to offer a less restrictive data filling in the XML file and to simplify the post-processing step for OMF, we have to work on sorted data. However, the XML standard provides only the display of these sorted values and no further processing is possible.

That is why many manufacturers have compensated this issue by adding new extensions in their XSLT processors while remaining compatible with the standard set of W3C. In our case, thanks to an intermediate variable which stores the result of the *sort* command in a result tree fragment (Rich Text Format (RTF)), we can convert the sorted RTF to a node-set on which we can process information as if we were in the real XML tree.

To support these extensions, the XSLT processor has to be EXSLT-aware. To the best of our knowledge, we can find the Saxon processor (professional/enterprise edition, not the free home edition) [oS10] and the free Xalan processor [xal10] both are built in Java. We finally used the latter.

```
<xsl:variable name="sortedChildren">
   <xsl:for-each select="child::node()">
      <xsl:sort select="@at" data-type="number" order="ascending"/>
      <xsl:sort select="local-name(.)" data-type="text" order="ascending"/>
      <xsl:copy-of select="."/>
   </xsl:for-each>
</xsl:variable>
```

Figure 5.22: Example sorted child

First data are sorted in chronological order and then in alphabetical order.
Second step consisted in putting these results in a variable ('sortedChildren').

```
<xsl:for-each select="exsl:node-set($sortedChildren)/*">
  <xsl:variable name="nodeSrcUsingTraffic" select="document('6nodes.xml')/
  <xsl:variable name="nodeDstUsingTraffic" select="document('6nodes.xml')/
  <xsl:variable name="nbrPattern" select="count(document('6nodes.xml')/sce
  <xsl:variable name="timeNow" select="@at"/>
  <xsl:choose>
    <xsl:when test="local-name(.)='startTraffic'">
```

Figure 5.23: Example sorted child

Third step was the conversion of the variable, which contained a tree, to a node-set. This way, we were able to work on data sorted by time and name [XML10].

## 5.12 XSL Transformation process

To enable the transformation, first download the XSLT Xalan processor (*xalan−j_2_7_1−bin-2jars.tar.gz*) on the website from the Apache foundation
http://xml.apache.org/xalan-j/downloads.html.
This processor, written in Java, has the big advantage to be multi-platforms and to be free. Moreover, it can be run by command lines, like a script. Then unzip the file in the directory of your choice. You will have to first import some libraries in your class path before running the XSLT processor.

- Under Windows, you can create an automatic script which takes 3 arguments:

1. xml file to transform
2. txt file for the output
3. xsl file for the transformation rules

```
java -classpath D:\xalan-j_2_7_1\xalan.jar;D:\xalan-j_2_7_1\xercesImpl.jar;
    D:\xalan-j_2_7_1\xmlapis.
jar;D:\xalan-j_2_7_1\serializer.jar;D:\xalan-j_2_7_1\samples\xalansamples.
    jar
org.apache.xalan.xslt.Process -IN %1 -OUT %2 -XSL %3
```

- Under Linux: you can create an automatic script which takes as argument:

```
java -jar xalan.jar -IN myXML.xml -XSL myXSL.xsl -OUT myResult.txt
```

## 5.13 Methodology

No scientific methodology has been produced or defined on how to add the consideration of a new network tool to a generic description language. Therefore, we established and proposed a methodology in four steps to include the management of this tool.

- Establish a first structure of a network description (if the generic network description does not exist yet); For example, define a "topology" part, "traffic" parts and "event" parts.
- Considering a concept $c$ and a set of the existing concepts $soc$ exploited in the XSD three of our tool. We analyse a new network tool and we discover a concept $c$. Should we include the concept $c$ in $soc$ (**s**et **o**f existing **c**oncepts)?
  - If $c$ has no default value and $c \notin soc$ : add this concept in the XSD with no default value;
  - If $c$ has no default value and $c \in soc$ with no default value in soc: $c$ is already defined and mandatory in the XSD;
  - If $c$ has no default value and $c \in soc$ with a default value in soc : $c$ is already defined in the XSD. Drop the default value already defined of this concept to force the user to specify a new one;
  - If $c$ has a default value and $c \notin soc$ : do not add $c$. The default value of the network tools will be used.
  - If $c$ has a default value and $c \in soc$ with no default value in $soc$ : $c$ is already defined using no default value. Keep this state to force the user to specify a new one;
  - If $c$ has a default value and $c \in soc$ with a default value in $soc$ : $c$ is already defined using a default value. If the two default values are the same, keep the default value to this concept $c$, else remove this default value to force the user to specify a new one.
- Create the XSL translation;
- Generate and test to remove all the errors.

The backwards compatibility due to these changes is not maintained since our tool has been designed solely to always generate experience descriptions for both platforms. Indeed, for the proper functioning of experiments on the two platforms, it is compulsory to specify the parameters needed for each tool. For example, in order to generate two experiment descriptions from a single scenario, as OMF is using real interface, the IP-addresses of network interfaces are required while this concept is absent in NS2. If we had chosen to make the specific settings not mandatory, then the user would not have known whether s/he had or not to give a value to these parameters in order to perform an experiment.

Nevertheless, if the user still wants to use our tool to generate a description for only one platform, s/he has to fill in all the required items, even if they are not used by the destination platform.

In the case of the IP-address example, it might be possible to construct a method for the automatic assignment of IP-addresses to the nodes depending on the network topology described. We have restricted the functionalities compared to the objective that we set out initially (see Section 1.1.1).

## 5.14 Contribution

Our contribution was to reshape the XSD (and thus, the XML) to handle the wireless network. This means to add elements, remove the unused ones and update some of them to make an attribute more explicit for its use. We have done many tests on WILE-E to discover some bugs that we have taken into account. The tool we created allows designing experiments for NS2 and OMF and was tested in Naples. Finally we proposed a methodology to add new network platform to an existing network description tool.

# Chapter 6

# Limits and Perspectives

In this chapter, we analyze the limits of our tool and consider the possible perspectives of this project.

From a technical point of view (the automatic generation of experience), our tool is currently targeted for the platforms NS2 and OMF. The user describes a scenario of experiment in XML language, and our tool automatically converts it to make the scenario usable on different platforms. Other languages may be included following the methodology we have described (see Section 5.13).

It is quite easy to add new features like a routing algorithm or a new type of traffic generator. The XML representation was indeed well designed for this purpose and to this kind of case. In fact, taking into account AODV has required only minor changes in our implementation.

Moreover, even if our tool is divided into several parts (XML, XSD and XSL), it does not use different programming languages but only one which is based on XML, reducing therefore the effort of learning.

Currently, the XML translator does not have a graphic interface, the user has to fill the fields provided in by hand. From a cognitive point of view, a GUI would improve the usability and the ease of our tool, while reducing the number of possible errors due to manual handling.

As seen in Section .1.4, in our tool the temporal events have been encoded in an identical way to the one adopted by NS2 in contrast to OMF. The latter requires a computation in order to express these events. It turns out that the complexity in the XSL to support a new type of events increases with the number of kinds of events you add.

What our tool does not: if the user describes a scenario incorrectly (e.g., using a network card that does not exist in reality) using a syntax that complies with the structures authorized by our language, then there will be no error in our application as it only plays the role of compiler and ignores how the operating platforms work.

Concerning the results of an experiment, a logical extension of our tool would be the establishment of an automatic comparison of results generated by the platforms. We should consider both output formats and their configuration to make the comparison possible, which is far from being obvious.

Another interesting perspective might help to overcome the problem of backward compatibility (see Section 5.13) of our tools. So far, our approach takes into account the common characteristics of all platforms, with the exception of specific binding parameters of each of them.
Two points have attracted our attention:
Firstly, the user wishing to create an experiment for a single platform should specify certain irrelevant fields (e.g., IP-address for the simulator). Secondly, the specific features of the platform have been reduced.

Our idea, which we briefly introduce below, could counteract these obstacles. In a few words, it consists in starting from feature diagrams to generate XSD files tailored to the user's request (Figure 6.1).

The concept of feature diagrams is quite easy to understand: it is a visual notation which lists all the configuration possibilities of a product (it might be something else) in the form of an "and-or" tree starting from a general root. The features are represented as the nodes of this tree and each of them must be either mandatory (filled circle) or optional (empty circle). We can also specify that only one

child node can be instantiated by using alternatives.

Feature diagrams are currently exploited, for instance, in the mobile phone product lines to list all possible configurations of the devices.

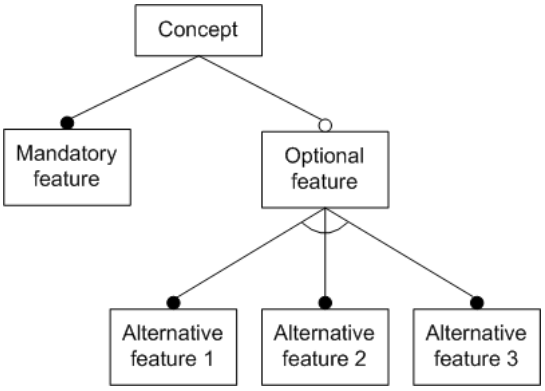



Figure 6.1: A feature diagram denotes a combination of features [Wei10]

In our case, one could imagine that it describes the structures allowed in our XML scenario of experiment description. For example, if the user wishes to only use NS2, then the sub-tree of the node representing this concept will list all the mandatory and optional features to use. Similarly, if the user wishes to use a combination of NS2 and OMF, the allowing structures will be listed in other tree nodes. Figure 6.2 shows an example of what such an approach could look like.

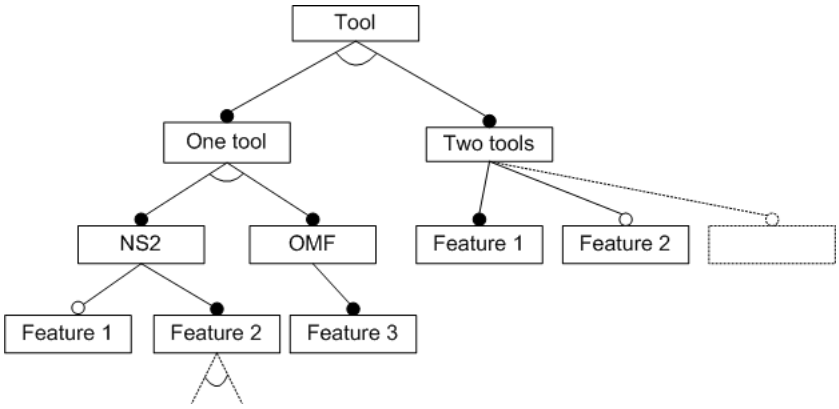



Figure 6.2: Idea of our approach

Once this diagram is modelled, the feature diagram will be translated into XSD files reflecting the authorized features, used in XML.

Although this idea is still incipient, we think it could solve the problem of backward compatibility requiring an additional step in the development of our tool.

From a societal point of view, we could assert that too often, only the simulators are used to validate and study the applications developed by researchers. As we have seen, these simulators are an abstraction of reality that just model some aspects that seem important to the simulator designers.

But reality is much more complex than what can be said and seen from it. This simplified complexity in favour of a better control of the world by the humans may truncate the results. Admittedly, simulations are for instance useful to study phenomena not directly accessible to humans (e.g., black holes in outer space), but one must really be aware that the results are obtained from a very simplified replica of the real world. Indeed, designers know from the world only the part they can perceive from it but the real world can not be reduced to these few observations. So, the implementation of the simulator contains only what man knows about the world, not the world as it is and as complex as it can be.

Therefore, based on this principle, the results obtained by the simulation must be questioned and validated by other means closer to reality. Our project in the context of this goal, thanks to the capacity

of using a testbed, can produce some results closer to reality. However, we are aware that the real tests are performed in laboratories and the real conditions are not always reproduced.

# Part II

# Reputation Protocol

# Chapter 7

# State of the art

With the presentation of wireless networks, this chapter provides the foundations for understanding the background of our second goal.

## 7.1 Types of networks

### 7.1.1 Mesh Network

The Mesh Network is a network topology without any central hierarchy wherein the hosts are interconnected. This topology looks like a fishing net, unlike the wireless classical centralized solutions in which nodes are connected with a single base-station.

To reach its destination peer, the traffic can jump from node to node until it succeeds in reaching it. To make the multiple hops possible, each node acts as an independent router, while keeping its feature of independence with respect for the other nodes: a node in Mesh Network does not care whether it is interconnected with another network or not.

As shown on the Figure 7.1, there are at least two different paths to reach each node of this infrastructure. As a result, the network may typically be very reliable, as there is often more than one way between a source and a destination.

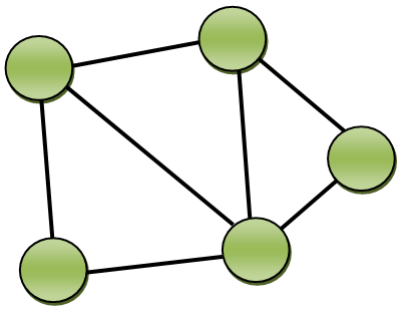


Figure 7.1: Example of mesh network

The great reliability offered by the decentralised architecture (avoid the single point of failure) is a "self-healing" feature. The network can still operate when one node breaks down or a connection goes bad because the traffic takes simply another available path.

Most of the time, mesh networks will be exploited for a wireless connectivity as explained in the next point.

### 7.1.2 Wireless Mesh Network

Wireless Mesh Network (WMN) [JS, IFAab04] is a wireless network based on a mesh topology (see previous section 7.1.1). WMN delivers wireless services by more and more deployments on campus,

metropolitan area, company networking and neighborhood Internet access, etc. This technology, directly in competition with broadband ones like cable, xDigital Subscriber Line (DSL), Internet satellite access, is a non-negligible alternative because of its advantages. It allows deploying a low-cost solution in areas where wired connections would be too expensive to make up.

The success of this technology is probably depending of several factors. The possibility to use low cost conventional devices like PDAs, access points to connect or extend the existing network makes CapEx (Capital Expenditure) very low. Moreover the technology can be set up incrementally by adding nodes progressively, at the desired frequency. A mesh network is reliable and offers redundancy and by adding some new nodes, the reliability and the network coverage grow up.

WMNs are made up from mesh routers and mesh clients, where mesh routers have a minimal mobility and form the backbone. Figure 7.2 shows a situation wherein users (small squares, in grey) provide Internet access with an important coverage area through the use of seven gateways (large rectangles, in red) to connect to the Internet.

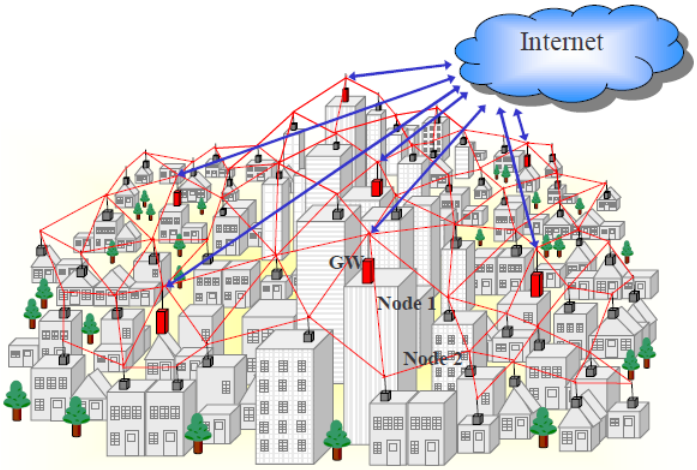



Figure 7.2: wireless mesh network deployed in a city [JS]

Mesh clients, usually laptops, cell phones, desktops, PDA's, or smartphones can be either stationary or mobile. Depending of the type of WMN (Infrastructure/Backbone WMNs, Client WMNs or Hybrid WMNs), client themselves can provide peer-to-peer networks among them without the need of any mesh router at the condition that they act as routers to forward traffic. Figure 7.3 shows an example of hybrid WMNs : mesh clients can communicate among them directly through their own network (at the bottom of the picture) without using the mesh router, or they can use the existing fixed infrastructure in order to access the Internet or also other mesh clients.


Figure 7.3: Hybrid wireless mesh network [JS]

The mesh routers, the second type of components, are often fixed nodes and provide the backbone infrastructure by means of routing protocols. Most of the time, the traffic from and to the gateways (often connected to Internet) is forwarded by these devices.

If mesh routers have the bridging functions, then the conventional clients equipped with the same radio technology as the routers can directly communicate with them, otherwise they have to use a wired technology. To avoid interferences and improve bandwidth at the router level, one type of radio is usually used for the infrastructure and another type for the clients. These networks can be interconnected with other ones like Internet or IEEE 802.11 thanks to the help of gateways and bridging functions in mesh routers.

Concerning the reliability of WMNs, if a node can no longer operate, it does not prevent the rest of the group from communicating together by using new network paths which have been calculated to send traffic through one or more intermediate nodes. The self−healing as well as the set up of a new node reconfigures the network node in a fully automatic way.

The wireless mesh network is a specialization of the ad-hoc network and thus routing protocols from the latter can be exploited in a WMN deployment. It offers a more structured and permanent environment than in ad-hoc where the communications are generally user to user and not fixed to any kind of infrastructure. Due to the high interconnectivity of the components, the routing protocol will often use a complex process to compute its routing table.

### 7.1.3 Routing protocol in WMN

Since the WMN is a type of ad-hoc network, it does not own routing protocols and uses those from ad-hoc networks. For greater efficiency, they need to be adapted to ensure performance and safety.

The mobile routing protocols are generally divided into two classes according to their way of discovering the network and establishing their routing table: proactive and reactive.

A proactive protocol knows all the network topology at any time, i.e., which node is present and how to reach it. This protocol creates and maintains a path to all hosts on the network. On the one hand, this principle enables sending traffic in a fairly fast way since it has all the necessary information concerning the destination. On the other hand, it requires a high bandwidth and a lot of information from the routing table. The proactive routing protocols include: OLSR, DSDV.

A reactive protocol does not know the network topology, i.e., it must decide which way to take to access a distant node before carrying traffic to this node. Therefore, this protocol adds an entry in the

routing table when it must contact this node and after a time of inactivity, the entry is deleted. This principle avoids spending bandwidth needlessly by the fact that it transfers only the minimal information strictly necessary to reach a node. Indeed, its routing table has no superfluous entries, which will perhaps never be used.

Unfortunately, this technique introduces a latency of several seconds before the transmission of data, time during which the resolution of the route and its adding in the routing table takes place. Reactive routing protocols include: AODV, Dynamic Source Routing (DSR).

## 7.2 Network security requirements

### 7.2.1 Contextualization

Wireless networks must deal with several types of attack different from the ones in wired environments. Due to the over-the-air propagation of packets, the traffic is not only received by the destination, but also by the other wireless devices within the transmission range. A malicious user, who can get an access quite easily to this infrastructure, has a lot of opportunities to compromise it.

In a wireless network, with the current existing security mechanisms, it is possible to achieve the goals of Confidentiality, Integrity, Availability (CIA). This means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification or destruction [Uni].

However, denials of service aimed at compromising the network infrastructure by preventing it from functioning properly still threaten the availability of services. An attacker who wants to break down or to corrupt its target will focus on important assets of the wireless network. Black hole and gray hole attacks, concepts coming from a survey of secure wireless ad-hoc routing [HP04], must be cited as examples regarding their dangerousness aimed at considerably reducing or interrupting the availability of a network service.

A black hole is viewed like the space-phenomenon which absorbs entities around it: a routing black hole on a malicious node attracts traffic, by distributing forged routing information, with the intention of dropping all the data packets. Since a lot of routing protocols base their metrics on a hop count to reach the destination, the malicious mechanism could be used to announce to other nodes its knowledge of very short distance routes to the nodes of the network.

Coming from this principle, a compromised device can be considered as grey hole if it only drops some specific packets. For instance, only routing packets would be forwarded and data packets would be discarded.

Among the available solutions (see [Oli07]) some require cooperation between all the components of a wireless environment to tackle these problems. Nodes collaborate by exchanging messages in an uninterrupted way and simultaneously they focus on the detection of suspicious behavior and actions to be taken to reduce the malicious activities.

### 7.2.2 Secure Routing Protocol for Wireless Mesh Networks

The problems of security concerns lead to the design of new trust-based modules, where the observation of neighbors' behavior contributes to attribute a value of trust to nodes, and thus their reliability index. Two of those mechanisms are Watchdog and Pathrater.

Watchdog is the module taking care of checking the successful packet forwarding from a node to another one. To act so, thanks to the over the air propagation of the data, this mechanism is capable to detect malicious behavior from nodes by listening on the same channel as its neighbors (every single node taking part in the network topology is set up with the same channel). It checks if its neighbor has correctly forwarded the data packets or not. It also detects the selfish behaviors without significant overhead. However, it requires a high storage, and itself needs to be secured against spoofing attacks which may badly affect the reputation systems depending on it. We will use this module as an extension to the reputation mechanism implemented in the routing protocol AODV.

Pathrater is a module combining the Watchdog information principle with the link reliability information. Nodes are accessible according to the metrics computed from Watchdog data. It thus refuses forwarding through nodes with misbehavior performance.

Those two modules are used to elaborate new theories e.g., REFACING or Cooperation Of Nodes: Fairness In Dynamic Ad-hoc NeTworks (CONFIDANT) [Oli07], to tackle the misbehavior a node could

have. This theory states the use of local and global reputations associated to a forwarding node behavior. This reputation model lengthens the hop-count-based metrics of the path which uses the malicious node, according to the trust into this node. As a consequence, the path through this node acting in a deviant way, will never be used.

# Chapter 8

# AODV-REX: A secure routing protocol

This chapter presents the routing protocol (in our case, AODV) and the extension to that routing protocol. This extension is inspired from the Ph.D. thesis of Francesco Oliviero.

We will first detail the routing protocol principle in order to understand how the underlying crafted reputation protocol works. However, it is important to keep in mind that the choice of the routing protocol is of no importance since the original idea was to prove that the reputation protocol is effectively correct. We chose AODV to be able to test the extension on the same routing protocol on simulators and testbeds.

The routing protocol is based on the collaboration nodes, and therefore keeps information of others behavior. This is the local information and global information. This system is based on the REFACING reputation model [Oli07], exploiting the local and global reputation.

Let us first introduce the routing principles and its basic principle.

## 8.1 AODV

### 8.1.1 Description

AODV is a routing protocol for mobile ad-hoc networking. Although initially scheduled for ad-hoc networks, it has been used in WMN's.

AODV allows wireless devices to exchange data by transmitting these ones through their neighbors if they cannot directly communicate. In this way, traffic is routed to the destination with the help of intermediate devices that support the rest of the routing process.

The mechanism exploited by AODV enables to save bandwidth: on the one hand, each node uses the information from the packet it captures, even without having made a request, to update its routing parameters. On the other hand, it creates and maintains a route only if the routing process needs it (see reactive protocol in Section 7.1.3).

Nevertheless, since the node does not know in advance the path to a destination where no data were delivered for a moment, a mechanism for creating a new route, taking a certain time, is required.

AODV also manages the fact that nodes do not contain any loops in their routing tables. This prevents from storing incorrect paths and spreading them through the network.

### 8.1.2 Principle

We will explain the main concepts of AODV as defined in the RFC3561 [Gro03], in order to provide the necessary tools to be able to understand Ad-hoc On-Demand Distance Vector routing - Uppsala University (AODV-UU) (from **U**ppsala **U**niversity), a C language implementation of AODV, which is presented in the next section.

AODV defines four types of UDP control messages, each containing routing information for a specific role: Route REQuest (RREQ) - RREP - Route ERRor (RERR) - Route REPly ACKnowledge (RREP-ACK).

**RREQ** Message generated for discovering a path to a destination when no route is known.

**RREP** Message generated by the destination or intermediate node as an answer to a RREQ message. It holds enough information to create a route to the destination.

**RERR** Error message which is sent to inform that a destination is unreachable.

**RREP-ACK** Message generated if an explicit request for an acknowledgement has been made. It follows the reception of a RREP message.

A neighbor is a node with which it is possible to communicate directly. At regular intervals, each node sends a hello message by broadcast. This one is a RREP message in which the contents of some fields are filled in order to distinguish them from normal RREP packets.

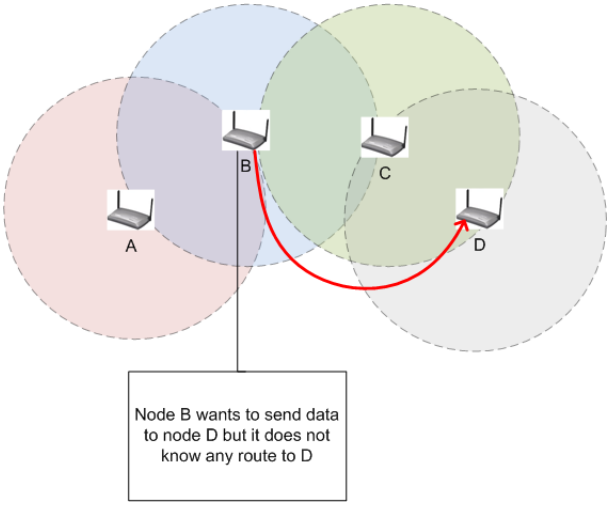



Figure 8.1: Where is node D?

**RREQ** A node wants to send a message to another one which is not a neighbor and does not know a route to it. Then to discover a new route, it broadcasts a RREQ message. On Figure 8.1, the network topology is built with four wireless nodes using the same transmission channel. The circle represents the communication range of their wireless materials. Therefore, each host can only transmit data directly to its neighbors.

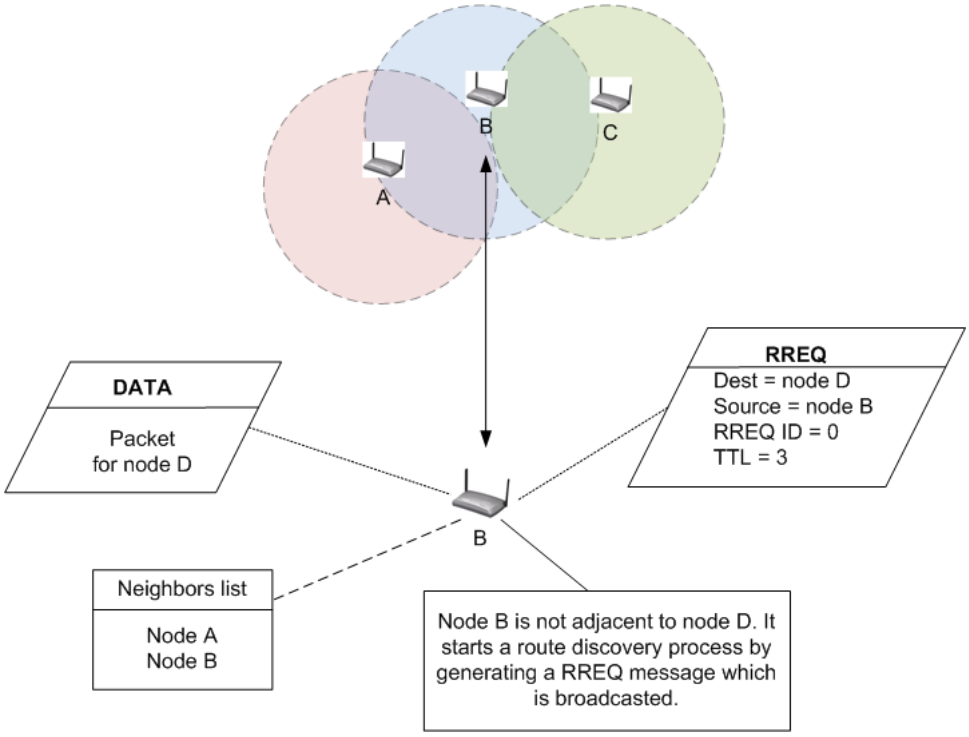



Figure 8.2: Route discovery process - node B generates a RREQ message

There are numerous fields in the message (see Section 5.1 from RFC3561 [Gro03]) but we will only discuss about the relevant ones which are the most important to understand the mechanism. The RREQ message contains information about the source, the destination, IP packet lifetime and a unique identifier which associated with the address of the sender can differentiate each RREQ across the network. On the example of Figure 8.2, node B wants to communicate with node D which is not a neighbor. It then begins the route discovery process by broadcasting a RREQ message.

After sending the route request, node B expects a RREP response within a certain delay. If the associated timer expires, it will generate a new RREQ with a longer packet lifetime and a new identifier number.

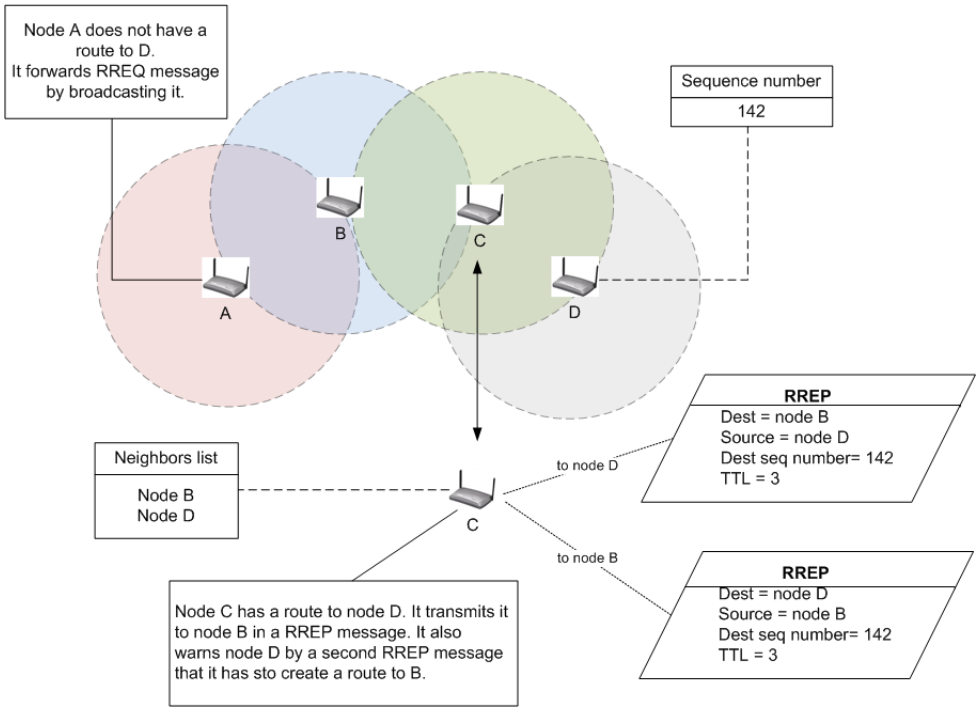



Figure 8.3: Node C has a fresh route to node D - node C generates a RREP message

Following the reception of a RREQ message, a node can react in three ways. In the first case, it does not know the requested destination and rebroadcasts this message to its neighbors. This RREQ is forwarded until its lifetime (Time To Live (TTL)) is less than 1. In the second case, either the node itself is the destination or is only an intermediate node with a route up-to-date to the destination. It then generates a RREP to the originator of the RREQ packet. In the last case by comparing the sequence numbers (presented in next point), the node knows the a path to the destination is not fresh enough and thus rebroadcasts this message to its neigbors.

**RREP** In figure 8.3, node C has a route to node D. It informs node B that it has the route and also tells node D that it will soon receive traffic. As node A does not know how to reach the requested destination it forwards the RREQ message by broadcast.

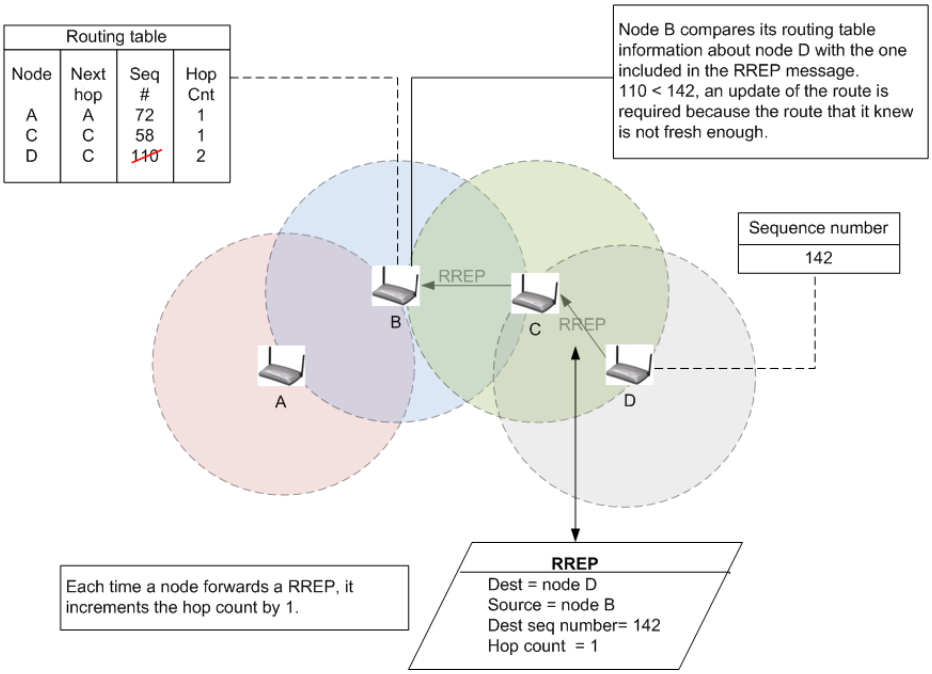



Figure 8.4: Node B compares the sequence number of its routing table with the one present in the RREP message

**Sequence numbers** They are used to compare the freshness of locally maintained information about their neighbors, with the actual state of their neighbors. At each emission of a control message from one of the four types, the host will increment a local sequence number. Each wireless device records the sequence number of the nodes with which they are interacting.

Since a higher sequence number means a fresher route, the node knows that it has to refresh the information in the routing table. How does it work? On reception of a control message, it compares the sequence number encapsulated in the packet with the last sequence number it knows about these nodes and updates its information if needed.

In Figure 8.4, node C that did not have a fresh route enough to node D has rebroadcasted the RREQ. Node D, directly concerned by this message, generates a RREP to node B. All the nodes that intercept this message update their routing tables if necessary (i.e., if the sequence number they know for a destination is inferior to the one specified in the AODV control message).

**RERR** An error message is generated to inform that a destination is not or no more reachable by using a route that would pass through the node at the origin of the message.

A broadcast of a RERR message can occur in different situations.

- A node detects that it can no longer communicate with a neighbor. It then invalidates all its routes using this node. The unreachable host is also reported by a RERR message broadcasted to the predecessors, i.e., those nodes which are using the node that detect the link break, to reach the unreachable destination.

  A node that receives a RERR for at-least one of these routes that will be invalidated, marks as invalid all its paths stating that node and forwards the AODV control message to its predecessor(s).

- A node receives a data packet that it is supposed to forward to a destination that it does not know. So, the sender thinks wrongly that the correct route to the destination passes through that node. Therefore, the node that got this data packet informs the sender by an error message.

In both cases, when a node receives such an error message, it removes all the routes that mention this node from its routing table. It then propagates this message to its predecessor(s).

## 8.2 Reputation EXtension - Mechanisms used by AODV-REX

In this part, we will consider the reputation mechanism principles, its propagation setting and how to calculate the reputation values. This is mainly a reformulation based on the article *A Reputation-based Metric for Secure Routing in Wireless Mesh Networks* [OR07].

First of all, let us introduce the important notions to understand the reputation model.

The goal of the mechanism is to detect a malicious node that does not correctly forward the traffic which goes through it and prevent future data from following this path. The reputation, which is expressed by a number between 0 and 1, indicates how much a node can be trusted by others through a history of its behavior when forwarding data packets. The reputation mechanism is based on three reputation values: the local reputation, the global reputation and the current reputation spread in the RREQ packets.

1. The **local reputation** is the one held by a given node about its neighbors. In this given node, each of its neighbor(s) is associated with a different local reputation, representing the reliability with which those nodes successfully forward the packet sent to them. The Watchdog module is in charge of updating this reputation.

2. The **global reputation** is based on the observations by the network nodes about a given one. It is a merging of all the reputations that the neighbors of this given node have calculated about it.

3. The **current reputation**, propagated through the network, is a merge of both local and global reputations that can be exploited to determine the real behavior of a node. In other words, it represents how much a node is trusted by the whole network plus how much each node trust that given node (only when that node is a neighbor). By merging the local and global reputations, this mechanism prevents a malicious node from precisely identifying which of its neighbors has begun to propagate a bad reputation about its behavior. By doing so, the deviant does not know how to react to stop this propagation.

### 8.2.1 Local reputation

We have to keep mind that each node has its own Watchdog module. The local reputation is kept updated by that module which ensures the smooth transfer of data carried by the neighboring node is done through the mechanism explained in the diagram of Figure 8.5.

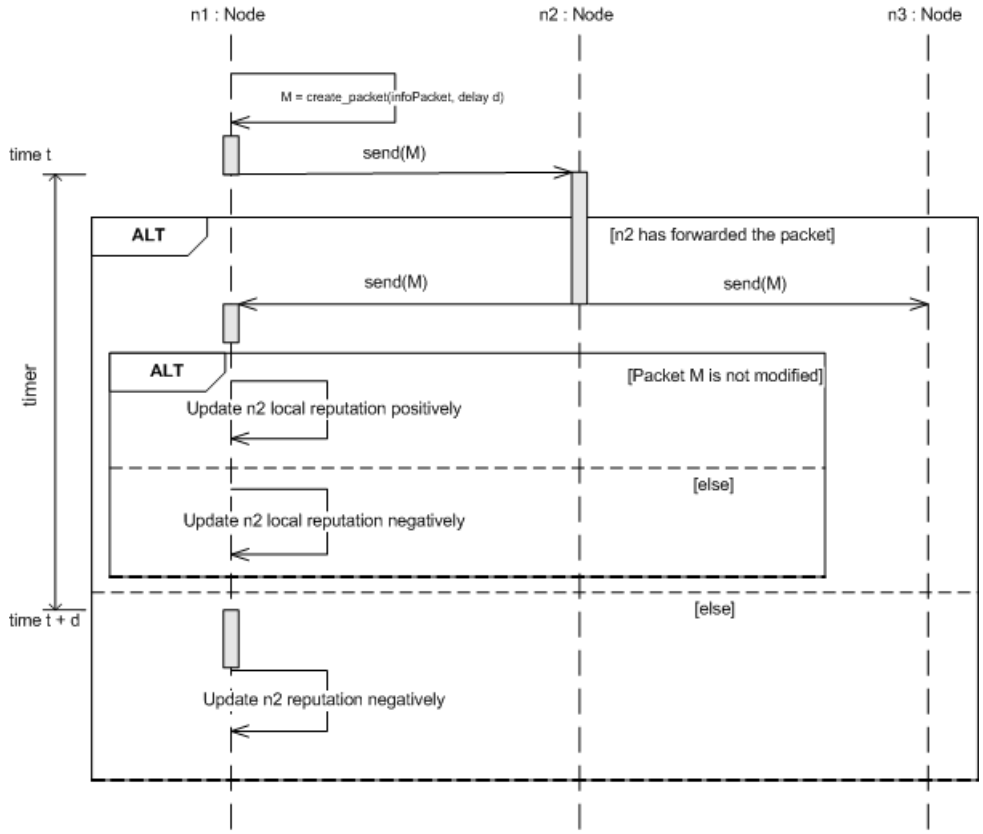



Figure 8.5: Watchdog process on node n1 (UML sequence diagram)

In our scenario, the node *n1* wants to send a data packet to the node *n3* which is not a direct neighbor. Following its routing table, it sends this message to the node *n2*, which is the next hop in the previously established route. The sender also starts a timer for a delay 'd' for this packet.

At the reception, the node *n2* can react in different ways:

- *n2* processes this message and forwards it to *n3* without altering the content. Node *n1* receives a copy of this message (due to wave propagation in a wireless environment), compares this with the previously sent packet to *n2*, confirms that it is the same content and assigns a positive local reputation to *n2*.
- *n2* processes this message, alters the content and forwards it. Node *n1* receives a copy of this message (due to the waves propagation in a wireless environment), compares this with the previously sent packet to *n2*, detects the changes of content and then assigns a negative local reputation to *n2*.
- *n2* processes this message but it never forwards it (or at least, *n1* does not hear it). The timer at node *n1* expires and it then assigns a negative local reputation to *n2*.

Watchdog updates a local reputation list containing the result of the last forwarded data packets. Thanks to this mechanism, we have a history for each neighbor at our disposal.

### 8.2.2 Global reputation and packet reputation

Upon receiving a message encapsulating a reputation value, a global reputation is calculated from this value and some local informations and this value is added to a second list containing a history of the last global reputations obtained about a node.

Just before sending a message, the reputation process takes place: by the combination of the two previous lists, a value representing the current reputation of a node, is calculated and then in turn encapsulated in the message for sending.

The last step of the reputation extension principles modifies the basic metrics (hop count) of the routing table to force data to avoid a detected malicious path: the current reputation is translated into a number which is added to the normal hop count. A bad trust results in a high number, while a good one tends to 0.

A direct consequence of this will be the lengthening of the path if it goes through unreliable nodes.

As the aim is to keep the choice of the shortest path, the paths including malicious nodes would not be considered as relevant (the path becoming too long).

By doing so, the process would "transform" the principle of the shortest path first (the source wants to reach the destination in a minimum number of hops) into the most trustworthy shortest path first (the source wants its packet to be handled by reliable nodes only). The underlying idea is to keep every node in the graph and not remove them from the moment they act deviously, especially if one of them is the single point to access the rest of the network.

### 8.2.3 Reputation model and Watchdog module

This reputation protocol is using the REFACING mechanism along with the Watchdog module.

This reputation extension is based on a new theory, REFACING that is organised in four layers. Let us detail those different layers.
(1) The first layer has a role of determining the existence of a relation between the nodes. If two nodes can interact, information from this layer will be provided to the second layer.
(2) The second layer will quantify the degree of interaction between the nodes. However, this level of communication between nodes has still no effect on the value of trust given to them.
(3) The third layer will then take care of the confidence nodes can have about each other. The trust value will determine how the nodes have interacted between themselves.
(4) Finally, the fourth layer will take care of the behavior history. It will observe how the interactions occurred in the past, and attribute a positive value if the node is reliable, and a negative one otherwise. All those interaction values will then be computed to give a general trust value about the other nodes. The sub-layers bring information about how the interaction can be treated. The more nodes are interacting, the more we can suppose they can trust each other.
In a practical way, this model includes the local reputation and the global reputation provided by the other nodes of the network to determine the reputation value spread between the nodes. This is done by the cooperation of the third and fourth layer.

### 8.2.4 Reputation calculation

The dissemination of reputation is done through the AODV process and is therefore based on its routing messages to propagate the reputation data. The following formulas are extracted from the thesis and scientific article written by Francesco Oliviero [Oli07] [OR07].

Let us introduce the notation describing the reputation calculation:

- $R_A[B](i)$ refers to the reputation of node B at node A at the $i^{th}$ iteration
- $R_A^G[C](i)$ represents the global reputation of C at node A at the $i^{th}$ iteration
- $\bar{R}_A^L[B]$ is the average of local reputation of B at node A
- $\bar{R}_A^G[B]$ stands for the average of the global reputation of B at node A
- $\bar{R}_A^\omega[B]$ represents the evaluation of reputation data received from node A about node B

The reputation computation starts when a node tries to send data and broadcasts the RREQ message. This message contains the reputation information which will be spread and mixed up with the local and global reputation every node stores about every other nodes. Let us define a scenario to illustrate the reputation calculation. Node C requests a route and therefore sends a RREQ message. Node B receives that packet and because it cannot reply to the request, inserts the reputation data it holds about node C and rebroadcasts the message. When node A receives the packet from node B and if node A is reputation-compliant (in our example, it is), node A will check the packet if it contains reputation data. If so, node A will take care to check the validity of this information, in case a deviant node would like

to ruin another node reputation (e.g., the malicious node B could transmit a falsified reputation about the node C). Node A will therefore weight this information (1) then compare it with the last reputation it knows about node C (2).

$$R_B^{\omega}[C](i) = R_A[B](i-1) * R_B[C](i) \ (1)$$
$$\Delta_{C_{AB}} = |R_A^G[C](i-1) - R_B^{\omega}[C](i)| \ (2)$$

After the check of validity, the global reputation of node C can be updated with

$$R_A^G[C](i) = \tfrac{1}{2}(1 + \Delta_{C_{AB}}) * R_A^G[C](i-1) + \tfrac{1}{2}(1 - \Delta_{C_{AB}}) * R_B^{\omega}[C](i) \ (3)$$

When a node is removed from topology, it can be reintegrated at the condition it has proven to have a recent not objectionable behavior. The local reputation (from direct neighbors observation) and global reputation are making this reintegration possible by the way of a *Weighted Moving Average ($\omega_{lr}$)* with

$$\bar{R}_A^x[B] = \omega_{lr} \sum_{i=1}^{N-1} (1-\omega_{lr})^{i-1} R_A^x[B](i) + (1-\omega_{lr})^{N-1} * R_A^x[B](N) \text{ where x} \in \{\text{G,L}\} \quad (4)$$

the average value of node B at node A weighted with its behavior history.

However, when a RREQ message is received or a Watchdog timer expires because the neighbor has not forwarded the packet in the given delay, we then need to update the reputation of that neighbor. The reputation is calculated with (4) and the variation from the last reputation value with (2). The formula is thus

$$R_A[B](i) = \bar{R}_A[B](i-1) + \Delta_{B_A}(i-1) \ (5)$$

To evaluate the $\Delta_{B_A}$ value, we will compare the distance between $\bar{R}_A^L[B]$ and $\bar{R}_A^G[B]$ to check if both values assent to each other with

$$\Delta_{B_A}^{LG} = |\bar{R}_A^L[B] - \bar{R}_A^G[B]| \ (6)$$

The determination of $\Delta_{B_A}^{LG}$ is determined in agreement with a given threshold. If the $\Delta_{B_A}^{LG}$ is lower than this threshold, global and local reputation agree. If it is not, they disagree . Let us see how to compute the value computing $\Delta_{B_A}$

$$\text{(7.a) } \Delta_{B_A} = \Gamma * (1 - \bar{R}_A[B]) * \frac{(1-\Delta_{B_A}^{LG})}{2} \text{ – Agreement}$$
$$\text{(7.b) } \Delta_{B_A} = -\Gamma * \bar{R}_A[B] * \frac{\Delta_{B_A}^{LG}}{2} \text{ – Disagreement}$$

A negative value is calculated in case of disagreement to reduce the reputation value, as node B shows a misbehavior. The $\Delta$ term is

$$\Gamma = \alpha * R_A[B] + \beta * ID + \omega * V_A^R[B] \quad (8)$$

where

- $\alpha + \beta + \omega = 1$ (9)
- $\alpha = 0.6; \beta = 0.2; \omega = 0.2$ (10)
- **ID** is the Interaction Degree between two nodes
- $V_A^R[B]$ is the Variance of reputation of node B at node A

Once the new reputation value is calculated, the average reputation and the variance (as described above) have to be updated with the formulas :

$$V_A^R[B] = \frac{N*V_A^R[B](i-1)+(R_A[B](i)-\bar{R}_A[B](i))*(R_A[B](i)-\bar{R}_A[B](i-1))}{N+1} \quad (11)$$
$$\bar{R}_A[B](i) = \bar{R}_A[B](i-1) + \frac{R_A[B](i)-\bar{R}_A[B](i-1)}{N+1} \quad (12)$$

### 8.2.5 Consequence on metrics and path selection

The path metrics in the network routing is based on the number of hops between a given node and its destination: this is the shortest path first route selection. The formula (5) is the base of the new metric calculation. The hop count in AODV is determined through the message RREP. The new solution will attribute a new metric model, switching from the shortest path first to the most shorted trustworthy path first. In the old model, when a RREP message is received, the hop count is always incremented by one. In the new model, it tends to increment the hop count according to the trust a node has in its neighbors. The more faith it can have in a neighbor, the lower the hop count value is. In the opposite case, the less confidence it has in a neighbor, the greater the value. This way, if a RREP message sequence number from node $b$ is greater than a the one from node $a$ when received at node $c$, while the reputation of node $b$ issmaller than $a$ then the selected path will the one passing though $b$.

The hop count is calculated thanks to a Reputation Metric

$$RM_{\bar{AB}} = \lfloor (1 - R_A[B]) * ND \rfloor \quad (13)$$

Figure 8.6:

where

- $R_A[B]$ is the reputation of node B at node A
- **ND** is the Network Diameter, the maximum network diameter defined by the AODV protocol

Depending on the $R_A[B]$ value, the Reputation Metric (13) one is either equal to the maximum network diameter ($R_A[B] = 0$) or is equal to 0 if the neighbor reputation is maximal.

*The approach based on the increment of the distance by the Reputation Metric (13) in the RREP message ensures an intrinsic security of the protocol: by adopting this mechanism a subverted node cannot modify the distance, since the RM is added by downstream node. Such node might modify the distance in the RREP message received, but anyway its reputation is reflected in the RM computed by downstream nodes. [OR07]*

# Chapter 9

# AODV-FUUREX

After explaining the main concepts of AODV, we will consider an open-source AODV routing protocol implementation on which we have grafted the reputation mechanism developed by F. Olivierio. We then give some explanations which we consider relevant to mention about FUUREX and the reputation theory.

## 9.1 AODV-UU

AODV-UU [Nor10a] is an implementation of the Ad hoc On-demand Distance Vector routing protocol which is compliant with the *IETF RFC 3561* and was created in Sweden at the Uppsala University, hence the UU-suffix.

Initially coded in C language to be run under Linux, AODV-UU is implemented as a user-space daemon with kernel components. A port to the simulator NS2 has been accomplished in recent releases of the implementation, and thus contains some C++ code. The version used in our project is the 0.9.6, published on 2010-05-29 on the SourceForge website [Nor10b].

If the reader wishes to benefit from implementation details, s/he can see the original source code in which we have added many comments respecting the Doxygen syntax [Hee10], or s/he can directly consult the documentation generated by Doxygen. These two types of information are appended to this document in computerized form. In addition, when we modified this implementation, we established several Unified Modelling Language (UML) activity diagrams to assist us in mastering the code. You can find these charts in appendix .2.

## 9.2 AODV-FUUREX: a modification of AODV-UU

Starting from the AODV-UU version, we implemented the reputation protocol as described in the reputation process. We called our new implementation 'AODV-FUUREX', which stands for AODV - Fundp Uppsala University REputation eXtenstion. Figure 9.1 show the starting output of our implementation.

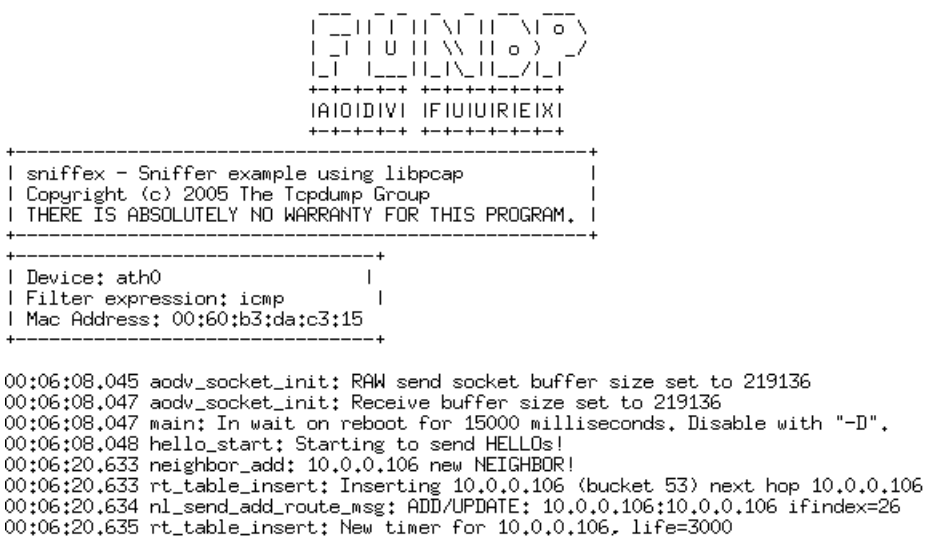



Figure 9.1: Launching of AODV-FUUREX

During the implementation of the security mechanism in AODV-UU, we faced several obstacles, some of them were more difficult to circumvent than others.

### 9.2.1 Released versions of AODV-UU

At the beginning of this project, the latest release of AODV-UU was published on December 8, 2007. As discussed in the introduction section, AODV-UU acts in user-space but also in kernel-space (module kaodv), which posed some problems. After dealing with all the problems related to the kernel version, a new version (0.9.6) was released on May 29, 2010 which fixed some bugs and made it compatible with the latest Linux kernel. We then transferred all our modifications onto the new one for the sake of compilation simplifications.

### 9.2.2 Problems encountered during development process

Since release 0.9.5 of AODV-UU, new kernels have been introduced in later versions of Linux. For this reason, the compilation with recent versions of Linux fails (kernel version $> 2.6.20$).

The kernel loaded on the most recent nodes of WILE-E is version 2.6.19, under which the compilation worked without problems. Considering the limited performance of such devices, no compiler (gcc in this case) is installed on the nodes. But to successfully compile, we simply have to install the kernel sources on the target machine (or Header). We learnt this information from the AODV-UU MakeFile file.

However, the older nodes of WILE-E have a 2.6.24 kernel version, which is too young compared to the existing implementation. Therefore, we got errors during the "make" compilation. The functions calls of the kernel being modified (e.g., changes in number of arguments), the compilation of the kernel component went wrong. No documentation in the code of AODV-UU and no maintenance were available. We contacted the author of the implementation and he wrote us he was not maintaining the sources. He gave us a link to what he has left about this work. However, the link provided us with no relevant information and the documents which were supposed to be hosted were not available. Fortunately, Giovanni Di Stasi (PhD student in Naples in charge of the WILE-E testbed) has developed a special patch to be applied before compiling, which has solved the problems of incompatibility (the new version does not require the patch anymore).

These old nodes also require a Cross Compiling, using specific tools to build the files (e.g., compiler-specific embedded chips), which cost us time to find the right configuration ("make arm").

As for the problems encountered on the testbed in Naples (see Section 9.2.5), it took between 10 and 15 minutes to run an experiment and collect the results. In this condition, the development and debugging of an algorithm is really not easy or feasible as it greatly reduces productivity.

In order to save time we considered the use of a network of virtual machines (VirtualBox / VmWare) in which we would have launched a Fundp Uppsala University REputation eXtenstion (FUUREX) wired

interface (although it is intended for wireless). We thought of making a network topology consisting of three nodes so that the traffic was forced to pass through an intermediary.

(A) <–> (B) <–> (C)

At node B, two network interfaces were necessary, but unfortunately the kernel module of AODV-UU prevents the algorithm from running simultaneously on two interfaces.

Following this problem, we considered the idea of using only one interface on each node but by adding filtering rules (for MAC addresses). With this technique node A and node C, being on the same LAN, could not communicate directly. This also proved to be a failure because the traffic did not move through the virtual wired network.

Finally, we opted to use our own equipment (laptops) as a solution for the development and testing to save time. Our supervisor, Prof. Schumacher, supplied us with additional computers and a version of Ubuntu 8.04 was installed on each of them. Then we applied the same topology as described above, using *iptables*[net10] (MAC filtering), which had worked perfectly in the early stages of the development.

The next stage was requiring to enable promiscuous mode on wireless network cards but this feature is only available on a small range of specific cards (requiring a special hardware, and specific drivers) and our laptops lacked this feature. Consequently, all functions related to packet sniffing, implemented for Watchdog, were due to be tested directly on the testbed in Naples.

WILE-E still being under development, we were forced us to adapt our deployment techniques during the project and find solutions to the encountered problems (e.g., changing the main file). We would like to stress the fact that the improvements made by Giovanni Di Stasi have significantly reduced the deployment time on WILE-E, allowing us to use it as a platform for development and testing.

### 9.2.3 The sniffer

A sniffer is a piece of program able to monitor packets on a network interface depending on different filters. This program was required to allow Watchdog to track the packets that have to be forwarded by the neighbors. So, the node running the sniffer can filter other packets as well as its own. However, to be able to capture those packets, a special hardware for the network card is required, so the given network interface can be set on promiscuous mode and effectively work (compared to others that can be set on promiscuous mode but no real result comes out). Thus, not every network interface can use this mode.

The sniffer we have implemented is designed in user-space only.

### 9.2.4 Path selection

We were faced with a problem concerning the route selection process according to the reputation values. As explained in the basic version of AODV (see Section 8.2), the creation of a new route has to be done by the sending of a RREQ and by the reception a RREP message. If several routes exist towards the destination, the originator accepts only the first RREQ message, while other packets which arrive subsequently are simply dropped. This ensures to select the fastest path to it, but not the most reliable.

Conversely, the challenge with the reputation mechanism is to select the shortest route, which is not necessarily the fastest. This has the consequence that, henceforth, the destination will no longer ignore the RREQ messages that have been rooted by different paths to it.

Two scenarios are possible: either the choice of the path is decided by the originator and the recipient will be notified, or it is done at the recipient end which will notify the originator. Each of both approaches has pros and cons.

The first idea, inspired by the work of Francesco Oliviero while deviating a bit from it, requires a three-phase process:

- The originator sends a RREQ by broadcast.
- The destination receives several RREQ and replies to each message by a RREP.
- The originator receives several RREP to which it adds the reputation information which is at its disposal. It finally chooses the safest route and informs the destination, which will have to expect to receive data via this path. This situation is shown in Figure 9.2.

The second idea consists in two phases:

- The originator sends a RREQ by broadcast. This packet, reaching an intermediate node, is directly modified with a new value of hop count according to the reputation the relay node has calculated about the neighbor that has sent this packet. Then it forwards the modified packet. Comparatively, in the first idea, the step of modifying the hop count was done during the RREP process when the packet is forwarded to the originator.
- The destination receives several RREP's to which it adds the reputation information at its disposal and chooses the best route. It has to only sent one RREP-ACK to the originator.

The advantage of doing so is that we simultaneously send the values of reputation and an updated metric in a single RREQ.

Wishing to stay closer to the idea of F. Oliviero, we opted for the first idea, knowing that the second one would have been possible.

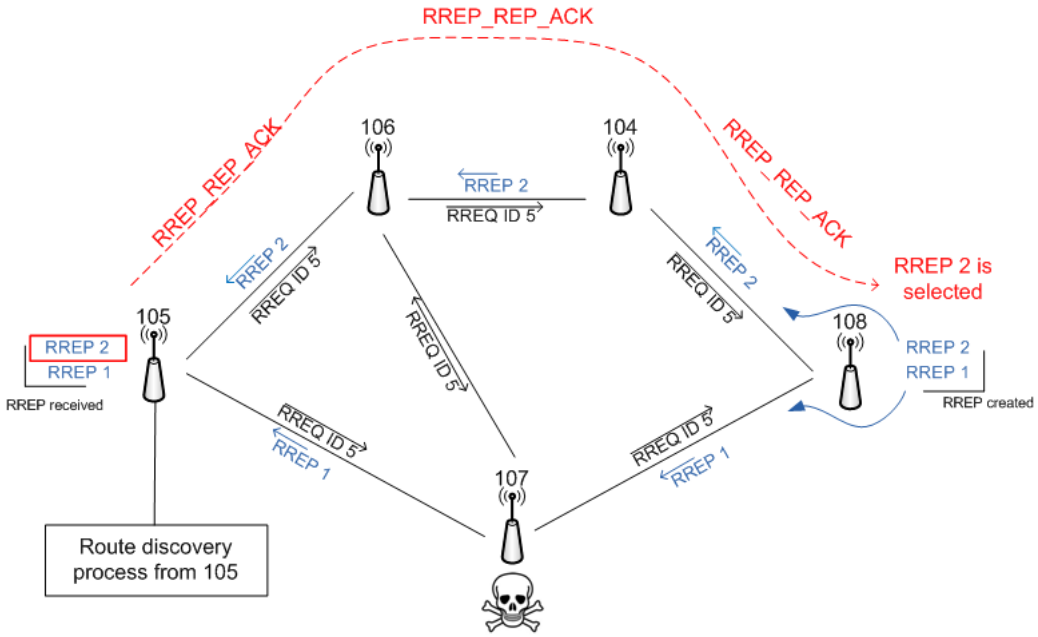



Figure 9.2: Route decision in 3 phases: (1) RREQ - (2) RREP - (3) RREP_REP_ACK

### 9.2.5 Theoritical limits

After analyzing the theoretical approach, we found several limits to the security model exposed.

#### Wireless channel

We first discovered a limit in the way nodes are supposed to interact. In order to allow the Watchdog module to capture packets, the nodes (the current node, its neighbor and the over-neighbor) need to use the same wireless band, and even the same channel. So, Watchdog can get proof from the neighbor the job is performed correctly. However, sometimes it is needed to assign dedicated channels between the couple of nodes. Thus, a node would have to use two channels to communicate with its neighbors.

If the fact that the whole or a part of the network is using the same channel is none of a problem for a determined type of job, it could be for others. The nodes would have to use an extraenous system, called *iptables*, to only allow traffic between certain given nodes, so the way followed by the packets is restricted to our designed topology under certain given constrictions. There are multiple ways to realize that from using *iptables*, blocking the traffic on an IP-based or a MAC-address-based method. But that reduces the whole use of the technology we currently have at our disposal.

### Border Router

In the case of a Border Router, whose location is at the border of the network, it is highly likely that the router will use two network interfaces : one for the wireless communication and the other one being is a wired interface. It would be better to provide a mechanism so the Watchdog instance of internal nodes is not active on the traffic passing by this Border Router. Either way, the Watchdog would listen and would expect the packets to be forwarded whereas they never will since they will be sent on the wired interface causing bad reputation forever for this node.

### Route updated with RREP

According to the reputation principles, the metric in the routing table has to be updated once a RREP message is received. If such a message is not generated, the malicious node B, whose task is to forward the data, can block/interrupt all the traffic passing through it without being worried about its new bad reputation. Indeed, until there is no RREP message, the routes of other nodes do not change! A mechanism forcing an update of this route (e.g., after a loss of x packets) should have been set up to avoid this weakness.

### Reverse route does not take the reputation into account

Let us consider a simple scenario where node 10.0.0.104 sends data to node 10.0.0.106 through node 10.0.0.105 ("called next hop"). Its hop count is 17.

Kernel IP routing table (Seen by A = 10.0.0.104)

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.0.0.106 | 10.0.0.105 | 255.255.255.0 | U | 17 | 0 | 0 | ath0 |
| 10.0.0.105 | 0.0.0.0 | 255.255.255.0 | U | 1 | 0 | 0 | ath0 |
| 0.0.0.0 | 192.168.1.1 | 0.0.0.0 | UG | 0 | 0 | 0 | ath0 |

In the meantime, node 106 decides to send data to node 104. Because of the prior existence of the route to 104 in its routing table, no route request process occurs. Node 106 will see node 104 with a metric of 2, independently of the reputation of node 105 seen by node 106.

This is made possible by the fact that the hop count in the routing table is updated regarding the reputation value only when there is the reception of a RREP.

Kernel IP routing table (Seen by C = 10.0.0.106)

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use | Iface |
|---|---|---|---|---|---|---|---|
| 10.0.0.104 | 10.0.0.105 | 255.255.255.0 | U | 2 | 0 | 0 | ath0 |
| 10.0.0.105 | 0.0.0.0 | 255.255.255.0 | U | 1 | 0 | 0 | ath0 |
| 0.0.0.0 | 192.168.1.1 | 0.0.0.0 | UG | 0 | 0 | 0 | ath0 |

Only the reception of a RREP message will force to modify the number of hops passing through node 105 towards node 104.

### Weaknesses in the mechanism

This section explains the weaknesses in the current reputation mechanism.

**Data forgery** : The information stored in the table of acknowledged packets (from or for Watchdog) includes only a few header fields of the IP packets. For performance reasons, storing all the data of a packet is not feasible.

If an attacker alters the content of the data in the packets while the traffic is still correctly forwarding the packets to the destination, the security mechanism in place will not detect any malicious activity.

The solution we would suggest is the calculation and the storage of a checksum computed from the data fields of such packets. However it remains to determine whether such a calculation would not adversely affect the overall performance of the device.

**TTL forgery** :
In the case of the TTL would be modified, there is no posibility to prevent the damages that could cause in the current implementation. So, the AODV message or the data packet would be simple dropped, causing a bad reputation to the node dropping it.

## 9.3 Methodology

In this section, we explain the methodology followed step by step in order to achieve the second objective (see Section 1.1.2)

- The first step is to refactor the existing comments in the original source code of AODV-UU to adapt it to the syntax of Doxygen [Hee10]. This brings a better visibility on the structures and methods used. It also allows faster access to information.
- Based on the theoretical aspects of the reputation mechanism, the second step consists in identifying the different moments of interaction between the mechanism and AODV in order to prepare the next action.
- We analyzed the code of AODV-UU to spot the places identified in the previous step. We then update the source code to include the mechanism of reputation.
- The development time ends by a debugging phase to obtain a valid implementation.
- Regarding the routing protocol exploitation, we define a scenario of networking experience which will be used in the NS2 simulator and in the testbed. Then, thanks to our tools developed to address the first goal of our thesis, we translate this scenario into files understandable by both platforms.
- We finish by running experiments on both platforms to collect results for a subsequent comparative study.

# Chapter 10

# Experimentation

## 10.1 AODV-UU vs AODV-FUUREX

We made a comparison between the AODV-UU and AODV-FUUREX implementations. We expected that our changes cause a more greedy use in terms of resources because we have only added features to the routing protocol. The results are displayed on the following graph (see Fig 10.1).

In order to perform that experiment, we created an XML description of the topology. This file can be found in its XML formal in the appendix, Section .4.

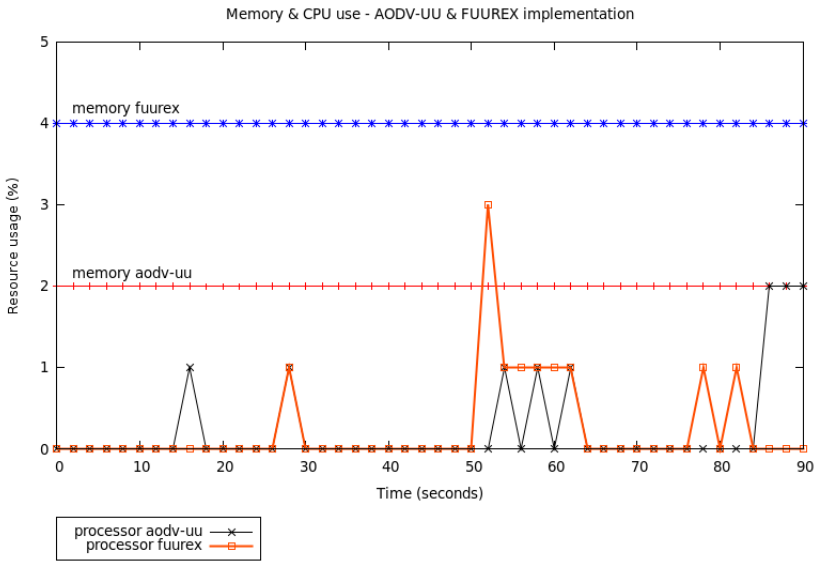



Figure 10.1: resource usage of AODV-UU and FUUREX

The experiment is launched with AODV-UU and with FUUREX afterwards. This was conducted following the same scenario by running a sequence of three pings. Several other experiments based on the same scenario confirm theses results.
The applications currently use a part of available physical memory, which is expressed on the graph by a percentage.

| **In the basic version of AODV-UU** | **In AODV-FUUREX** |
|---|---|
| the memory usage: 2% | the memory usage: 4% |
| CPU usage: from 0 to 2% | CPU usage: from 0 to 3% |

The difference in processor usage is negligible, even if there is a slightly higher usage in AODV-FUUREX.

On the other hand, the memory usage is twice as large as in FUUREX than in the basic version. This is explained by the structures added in our implementation in order to store information regarding the reputation and the Watchdog module.

## 10.2 Reputation results and consequences on the routing table

The following experimentation shows the evolution of the reputation of two nodes over time, which will be close to the value "1" for a node if it properly forwards data packets passing through it. Otherwise it will be less than "1" according to the importance of loss. We will interpret and justify those results with respect to the scenario we have established. The topology deployed on WILE-E testbed is shown in Fig 10.2. This topology includes the use of *iptables* to allow only the described links.

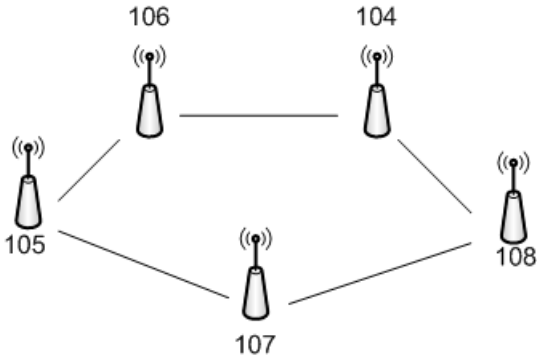



Figure 10.2: Experiment topology

Several bursts of data packets have been sent, preserving a break of a few seconds between each burst in order to start a new RREQ/RREP/RREP-REP-ACK process to refresh the current reputation of the neighbors.

In Fig 10.3, the vertical lines indicate a creation of an entry in the routing table to reach node 108. The cross on these lines denotes the route which has been chosen by AODV-FUUREX. The removal of routes is not shown on this diagram to preserve readability. The two other lines represent the reputation respectively of node 104 and 107, at node 105.

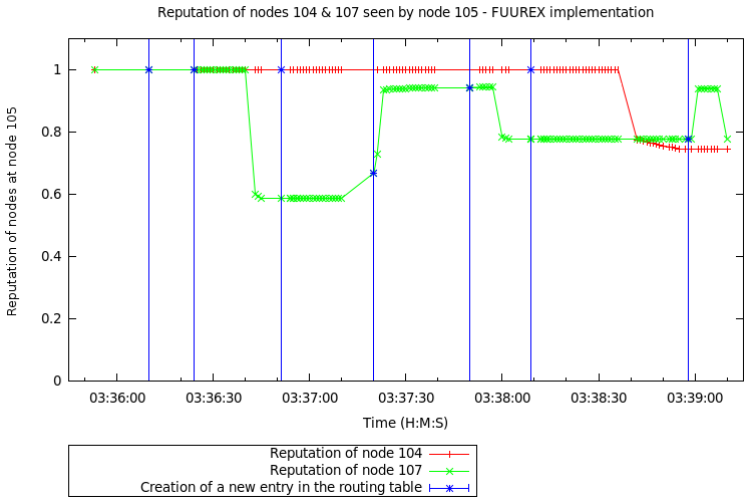



Figure 10.3: Reputations seen by node 105

**At 03:36:10** a route to node 104 and node 107 has been added to the kernel routing table.

**At 03:36:24** the first set of data packets has been sent from node 105 to node 108 through node 107, after the choice made by the reputation mechanism. Indeed, at the beginning, both reputations of nodes 104 and 107 are equal to one. Therefore, the selected path is the shortest route.

**At 03:36:51** a second set of data packets has been sent from node 105 to node 108 through node 104. Indeed, node 107 has lost some packets during the last forwards, resulting in a bad reputation and thus an increased hop count.

**At 03:37:20** node 108 has sent data to node 105 through node 107, despite the bad reputation attributed to it at node 105. This reaction is well-founded since in case of reverse route (as explained in Section 9.2.5), this is the originator (node 108) that selects the route. The reputation of node 107 seen by node 108 was not bad enough to avoid taking this next hop.

**At 03:37:50** a set of data packets has been sent from node 105 to node 108 through node 107. According to the metric formula (see Section 8.6, page 85), the number of hops taken into in account during the reputation process is the number of physical hops, plus a value derived from the reputation value. After performing the calculations, the value determined for node 107 was the same as for the node 104. In this case, the priority is given to the node that answered the most quickly.

**At 03:38:09** a set of data packets has been sent from node 105 to node 108 through node 104. The reputation of the latter is much better compared with the one of node 107.

**At 03:38:58** a set of data packets has been sent from node 105 to node 108 through node 107 reaching 8 hops (3+5, where 3 is the physical number of hops and 5 is the number of hops to add from the reputation process). The route through node 104 has not been selected because the reputation of this node has sharply plunged around 03:38:40 by dropping data packets.

The evolution of the number of hops in the kernel routing table of node 105 during this experiment, is shown in the figure 10.4.

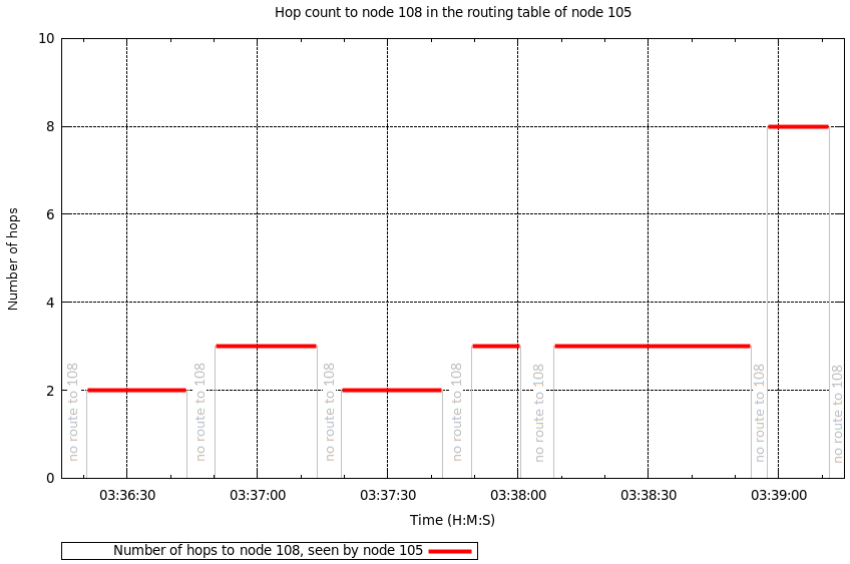



Figure 10.4: hops count to node 108 from node 105

# Chapter 11

# Limits and Perspectives

In this chapter, we will analyze the limits of the AODV-FUUREX implementation and provide hints to resolve these limitations.

The reputation mechanism improves the reliability of a wireless network by identifying malicious nodes. Our FUUREX implementation successfully performs this mechanism and works for the situations that are not include in next section. Aware that these situations are not yet supported we want to provide hints to overcome these obstacles.

We must also keep in mind that the reputation process solves only a part of the security problems specific to wireless networks (e.g, nothing prevents an attacker to deploy a wave jammer, thus preventing the transfer of information). Moreover, this approach requires all nodes in the infrastructure to communicate on the same channel, which can be an obstacle to its deployment. Finally, from the moment this solution can act positively on the security without significantly reducing the network performance, it is appropriate and even advisable to deploy it as soon as possible.

## 11.1 Unresolved routing issues

Based on Post-Ph.D. F.Oliviero's research in order to provide the efforts left to conclude, we nevertheless discovered during the last operating progression phases that new scenarios have not been considered.

The resolving time and their complexity to settle them being significant, we preferred to terminate our current implementation and to produce a working prototype being aware, in the meantime, that some issues need to be tackled. The routing table can not contain two entries to the same destination simultaneously, otherwise it will provoke an error at runtime.

This section addresses these issues and suggests solutions.

### 11.1.1 RREP propagation

- **Problem**
  In the situation depicted in the figure 11.1, node *a* wants to communicate with node *f*. Upon receiving the first RREQ, node *e* creates a reverse route to node *a* in its routing table . Shortly after, when receiving the second RREQ (coming from an another path), the route previously created, is not updated and the information of the second RREQ is lost at node *e*.

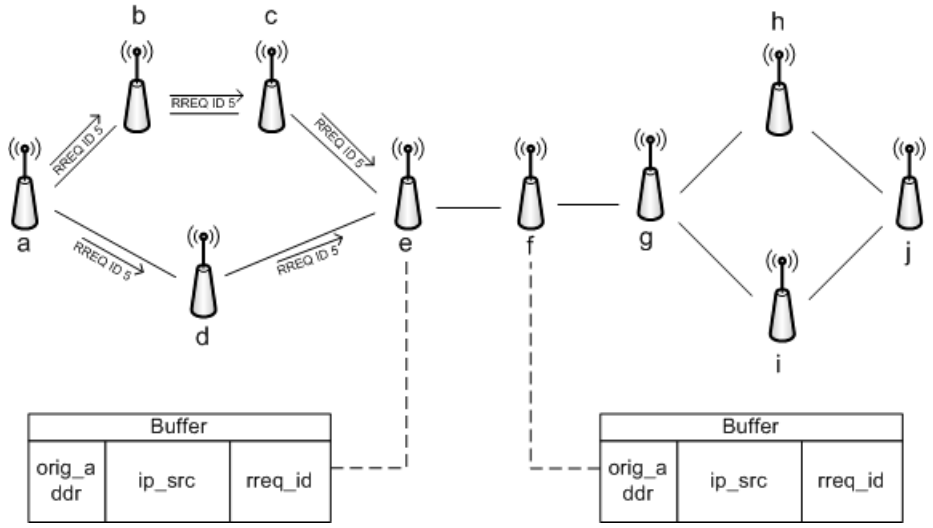



Figure 11.1: Scenario

On the return path, node $e$ will receive from node $f$ two RREP messages to forward to two different paths (i.e. to node $c$ and $d$). Unfortunately it knows only one route to node $a$, which entails that these two messages will be directed towards the same next hop. One of them will be deleted at that selected next hop node.

The routing table can not contain two entries to the same destination simultaneously, otherwise it will provoke an error at runtime.

- **Solution**
  - Change the implementation to allow storage of multiple entries to the same destination in the routing table and establish a way to remember if a RREP has already been transmitted by this route.

  or

  - Build a temporary structure where it would store the routing information (a sort of copy of the routing table) produced by these two RREQ messages. Anytime a use of one of these routes is required, it replaces the current entry in the routing table to the destination by the entry present in our temporary structure.

### 11.1.2 Single spot of juncture

- **Problem**
  The problem is related to the reception of multiple RREQ messages at several nodes in a row where one of them is the unique way to access the rest of the network. Let us consider Figure 11.1 again.

  Node $a$ needs a route to reach node $j$. It therefore starts to send a RREQ. The first message to reach node $e$ is coming from node $d$ since it is the shortest path to it.

  The packet is then inserted into the node $e$ local buffer. The bookkeeping information to keep tracks of the RREQ messages are: *Origin IP address, Source IP address & RREQ identifier.*

  The second RREQ message, forwarded through by nodes $b$ and $c$ is received. The packet is also stored, being different from the previous one by the *Source IP address* field.

  The packets continue their way up to node $f$. However, the stored information on this node cannot help any more to determine which packet was coming from which path. As a result, the second RREQ message received is erroneously dropped drop as considered as already received (due to the information present in the local buffer, already present in the basic AODV implementation).

  The current situation has found its limit.

- **Solution**
  The buffer should contain the different routes taken by the packets from the origin to the destination in order to fully qualify them and therefore identify them at node $f$ and the following ones. Node $j$ being the destination, the routing table can simply solve the problem. This solution implies the storage of IP addresses in the packets and their packets becoming fatter as the path gets longer.

### 11.1.3 Preventing RREQ cycles

- **Problem**
  The current implementation deals with the RREQ cycle problem. However, the way it currently operates is not optimal.

  When a node broadcasts a RREQ message to discover the route to communicate with the destination, all of its neighbors will receive this message. The receiving nodes which is not the destination nor do not know the route to the destination will rebroadcast the packet. The sender will then receive its own request and treat it as a normal request to forward.

  This happens because a node treats a packet since it does not have its information in its buffer (*Origin IP address, Source IP address & RREQ identifier*). Since the nodes do not buffer the RREQ messages they send, the message is sent three times on the link: Sender -> Next hop (Next hop stores the information) ; Next hop -> Sender (Sender stores the information) ; Sender -> Next hop (Next hop recognizes the packet and drops it).

- **Solution**
  - The packet sending request messages could record them into a sending buffer. By this way, when a RREQ message is received, the node will have to check both its sending and receiving buffers. If the information is in none of them, then it is a brand new message to handle.

  or

  - Insert the nodes IP addresses taken by the RREQ packets. So, when a node receives a RREQ message, it will seek into the IP addresses list and check if its own IP is in. If so, a cycle would be started and the node should not forward it. Otherwise, the packet is new and the RREQ message is on the right way to reach the destination.

# Chapter 12

# Thesis conclusion

At the end of our three months stay in Naples, we discovered a new world made of new technologies and new knowledge. Immersed into the research world, we discovered through the testbed and all the related softwares what is really happening when we talk about software testing.

We also had the opportunity to jump in an unknown environment and to live this period in the Italian culture which is quite different from ours. It was very rewarding to get in touch with people of various nationalities, including other Erasmus students coming from all over the world.

Throughout the development of this thesis, we became more and more familiar to the scientific approach by searching relevant articles and reading them, and by the direct or indirect contacts we had with other researchers. All these contributed to our attempt to apply scientific approach to our project. These documents allowed us to learn new knowledge related to our thesis by providing us, for example, how to model a network experiment in a right way. It also presented us with the rigor researchers need in their jobs.

In this challenge, we put into practice the knowledge and skills acquired both in a personal way as those acquired during our academic career. Our skills allowed us to adapt to new situations, analyze problems and seek the best way to sort them out.

Although we had to deal with problems related to the use of OMF on the Naples testbed, we believe that the majority of these problems is due to the version currently used. We insist on the fact that we could easily communicate with developers. Aware of the weakness of their documentation, they are now about to expand and to structure this part. For these reasons, we recommend that researchers should deploy this framework in their network test environment.

We also had the chance to delimit the meaning of our thesis and to understand the role of our work. Indeed, we were confirmed why the interaction between the testbed and the simulator is so important. So, we got aware that our subject was not solely a work, but that it was opening a door for further research.

In this thesis, we have been confronted with various areas such as programming in languages like XML and similars, C and Ruby. We have applied different concepts of network, exploited routing protocol and also modified some source code. This project has been really rewarding and we have acquired a better knowledge thanks to the notions we had to use.

One thing we have learnt for the rest of our lives is that we have to be careful when we have to schedule a high-risk project. Developping something in a language which is not fully mastered at the beginning of the project, is intricate : we had to learn and develop at the same time. Obviously, this has led to unexpected surprises, which we managed to overcome, though.

We finally finish this thesis on a positive note keeping excellent memories of our experience both in Belgium and in Naples and we would be pleased if it could prove useful to future researchers.

# Chapter 13

# Appendix

# .1 The modifications made to the XSD files from XDLWNS

This part describes the modifications we brought to the XDLWNS to match with our first thesis objective. As already defined in Section 5.7, here is the symbols meaning.

Figure 1: Sequence of element(s)

Figure 2: Choice among the children


Figure 3: Add comments/notes to each element


Figure 4: Adding multiplicities on different kinds of element

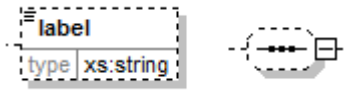



Figure 5: Different kind of optional element (multiplicity = 0..1)

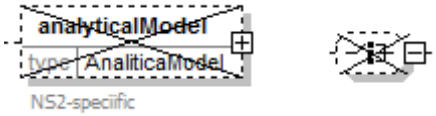
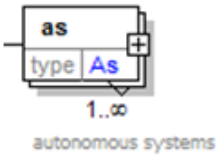



Figure 6: Different kinds of element that cannot be instantiated

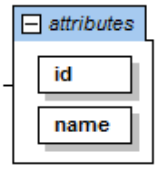
We defined a personal syntax to improve the understanding of the next section:

**UPDATE** An element has been updated
**ADD** An element or an attribute has been added
**DELETE** An element or an attribute has been deleted

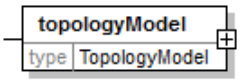
### .1.1 network Description (Fig 5.11, Fig 5.12)



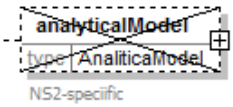

**UPDATE** Element *Autonomous System* has become mandatory because at least one logic group of nodes is needed.



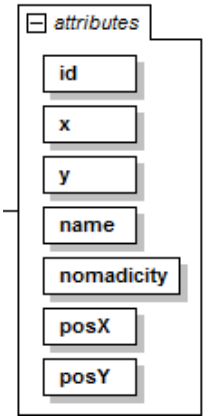

**ADD** attribute *name* added in the element 'Autonomous System' for a mnemonic help.



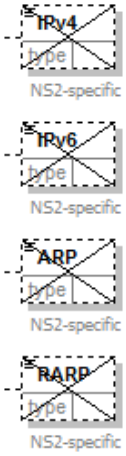

**UPDATE** Element *topologyModel* has become mandatory because it is the only common part describing a topology in OMF and in NS2.



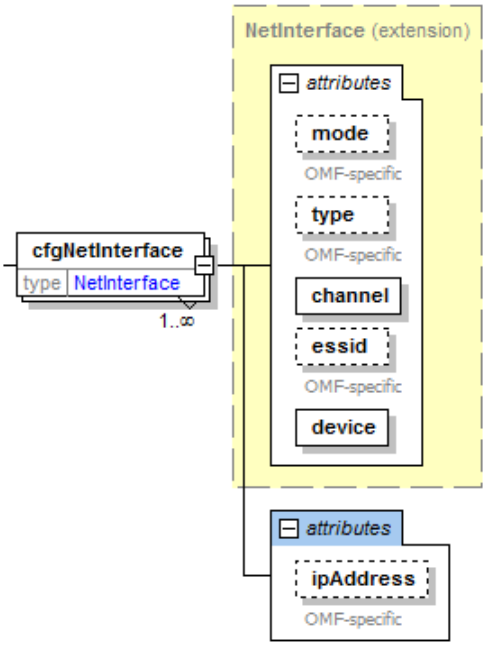

**DELETE** Element *analyticalModel* cannot be instantiated any more because it is only used in NS2.



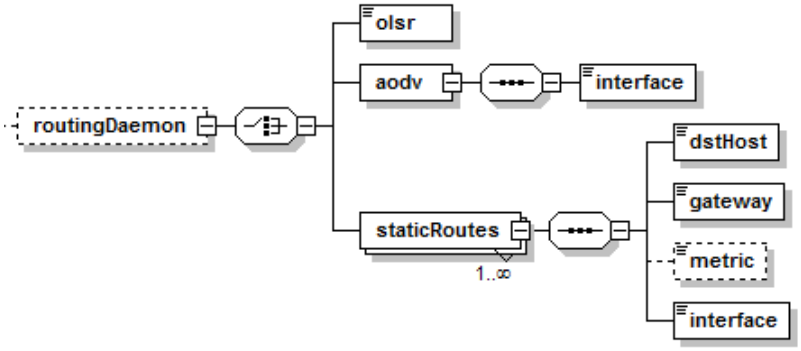

**UPDATE** Element *node* is modified. Nomadicity, posX, posY is needed in NS2.



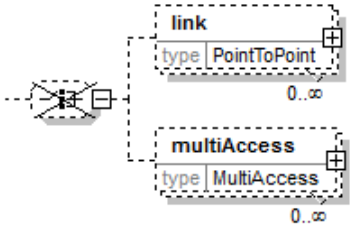

**DELETE** Some attributes of *protocol* element cannot be instantiated any more because it is not used in OMF.



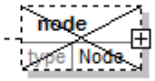

**ADD** Element *cfgNetInterface* is added as a child of the element node because it is needed in OMF. The attribute *channel* will be associated to an interface belonging to a specific node.



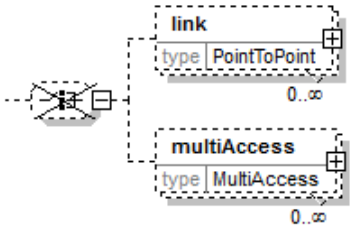

**ADD** Element *routingDaemon* (Fig 5.13) is added as a child of 'node' because it is a new functionality common to both platforms.




**DELETE** Element *link* and *multiAccess* in the element 'topologyModel' (Fig 5.11) cannot be instantiated any more because it is not needed in a wireless environment.

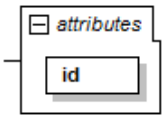



**DELETE** Element *node* in the element 'networkDescription' (Fig 5.11) cannot be instantiated because it was redundant (node is already declare elsewhere).

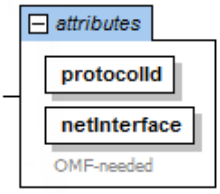



**DELETE** Element *link* and *multiAccess* in the element 'networkDescription' (Fig 5.11) cannot be instantiated any more because they are not needed in a wireless environment.

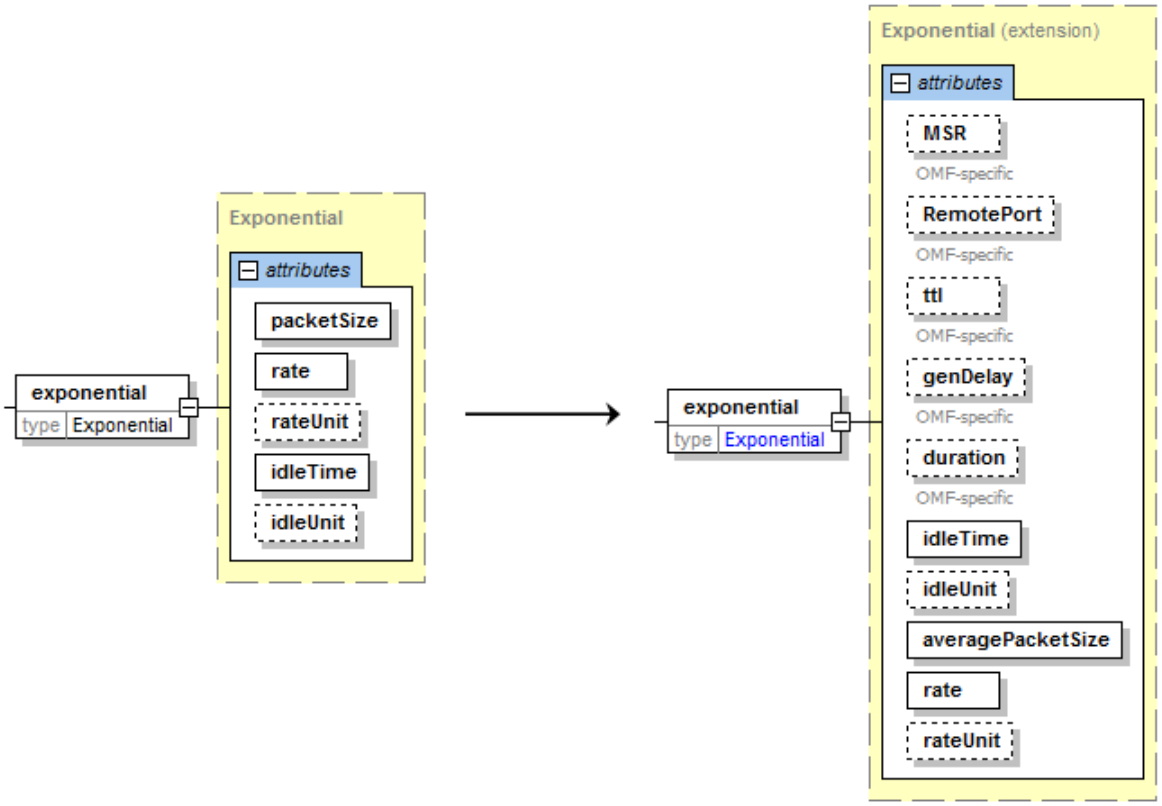
### .1.2 Traffic (Fig 5.15)



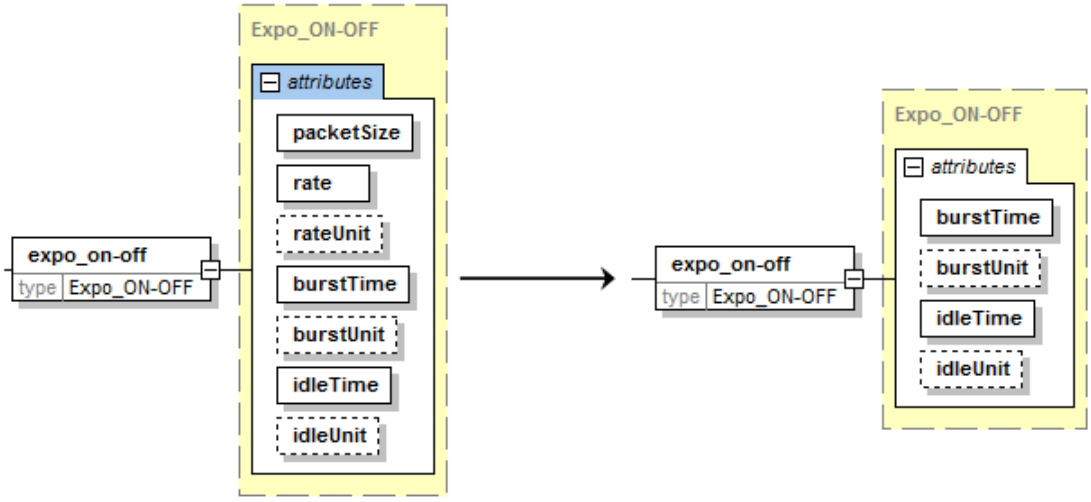

**DELETE** Attribute *startTime* and *stopTime* move into the element 'simulationCommand'.



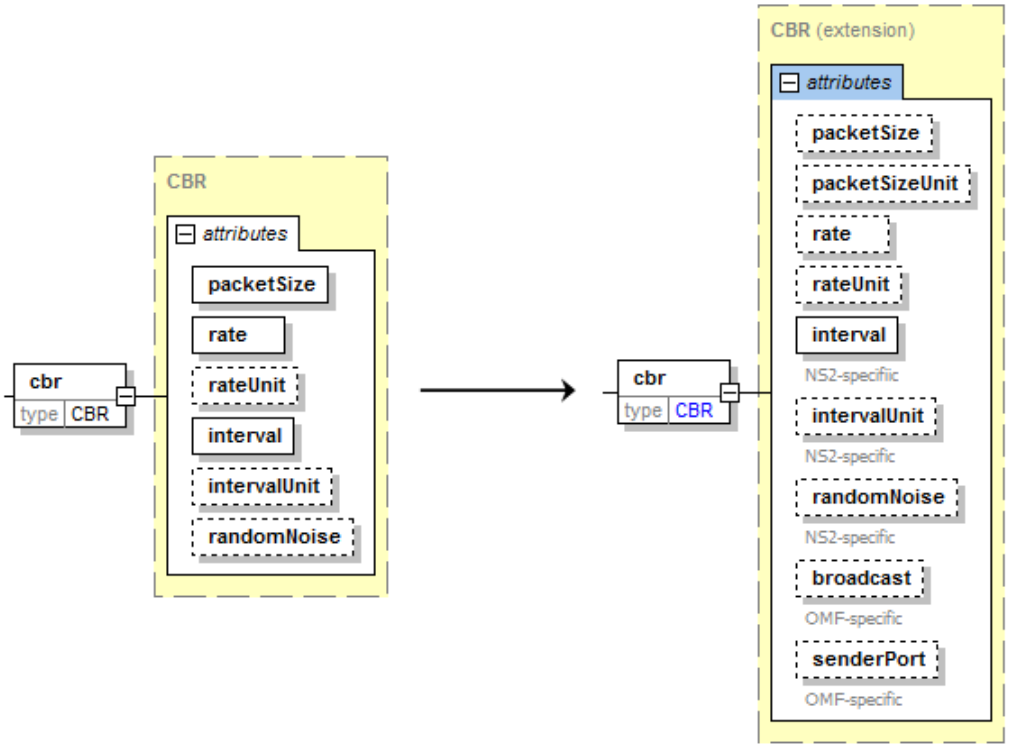

**ADD** Attribute *netInterface* in the elements 'src' and 'dst' in the pattern because it is needed in OMF.



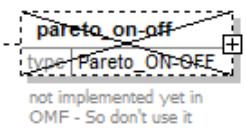

**UPDATE** Element *exponential* is modified (Fig 5.16). We add parameters which are needed to enable the traffic model.




**UPDATE** Element *expo_ ON–OFF* and its attributes (Fig 5.18). We delete parameters which are not needed to enable the traffic model.




**UPDATE** Element *CBR* and its attributes (Fig 5.17). We keep the rate and packetSize attributes, because they both allow finding the packets per second (Bandwidth = Size / Time and Packet per second = Rate / PacketSize).

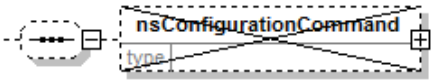



**DELETE** Element *pareto_ on–off* (Fig 5.16) is become optional because it is not available in OMF.

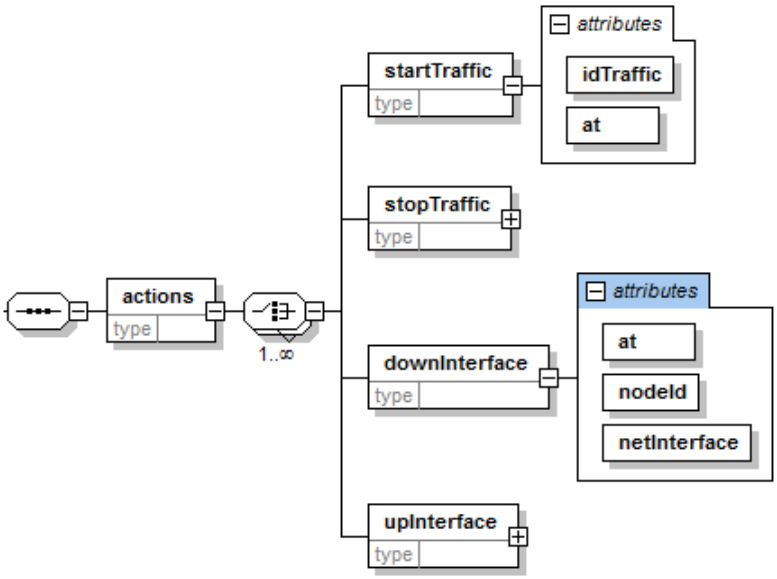
### .1.3 SimulationCommand (Fig 5.19)


**ADD** Element *authors* is added as recommended in LETSQoS project 5.5.2.


**ADD** Attributes *startLog* and *stopLog* are added because we need these parameters for our translation process.




**DELETE** Element *nsConfigurationCommand* has become optional because it is NS2 specific and not mandatory.




**UPDATE** Element *simulationEvent* was replaced by the element 'actions' which is more adapted to our situation.

### .1.4 Time representation in the XML scenario description

The time, in our XML, is expressed by giving only temporal points (at the $5^{th}$ second, at $10^{th}$ second). Unfortunately, OMF expresses the succession between two actions by the time that the node handler has to wait before the next action (wait 10s, wait 30s). We have decided to keep the first representation, used by NS2, which has to be computed for OMF. This is not an easy way but it is the only one we found. The problem is that each time we want to add a new type of action that can occur (like *start/stopApplications*), the calculations become more complicated and prone to error by the developer (see source code in annex).

The list below shows the dynamic events that we currently manage:

- The start time of the experiment
- The stop time of the experiment
- The start time of a traffic
- The stop time of that traffic
- The start time for the logging related to a source / sink traffic
- The stop time for the logging associated to that same source / sink traffic

### .1.5 Problem of granularity and concepts

All the nodes of a testbed in their range in a wireless configuration have the capacity to communicate, like in a mesh network. In OMF, to simulate an obstacle preventing communication between two nodes, we can specify a *removeLink* property whose effect is to add several firewall drop rules on nodes, so the nodes cannot communicate any more.

The only solution to reproduce this situation in NS2 is to place the wireless node out of the signal range of the others. The act to calculate the position of the node to reflect such a scenario is not obvious, especially in a complex network topology. That is the reason for which we think that this task should be performed by an appropriate graphical design solution, like in the Castadiva testbed (see Section 2.4.2).

In NS2, Autonomous System (AS) has no match in the OMF domain. Nevertheless, we maintain the concept of AS in the XSD but this is more to keep a logic description of the network since it has no effect on the generated files.

In NS2, the IP-address of the network interfaces is an unknown notion.

## .2 Packet processing in AODV-UU

This section describes some relevant parts of the inner workings of AODV-UU. In this way, it allows the reader to more easily understand the principles applied in the implementation by providing a good overview of its operation.

On the following UML activity diagrams, we made abstraction of the irrelevant details from our point of view. Therefore, if the reader wishes to benefit from implementation details, s/he can see the original source code in which we have added many comments respecting the syntax defined by Doxygen [Hee10], or s/he can directly consult the documentation generated by Doxygen. These two types of information are appended to this document in computerized form.

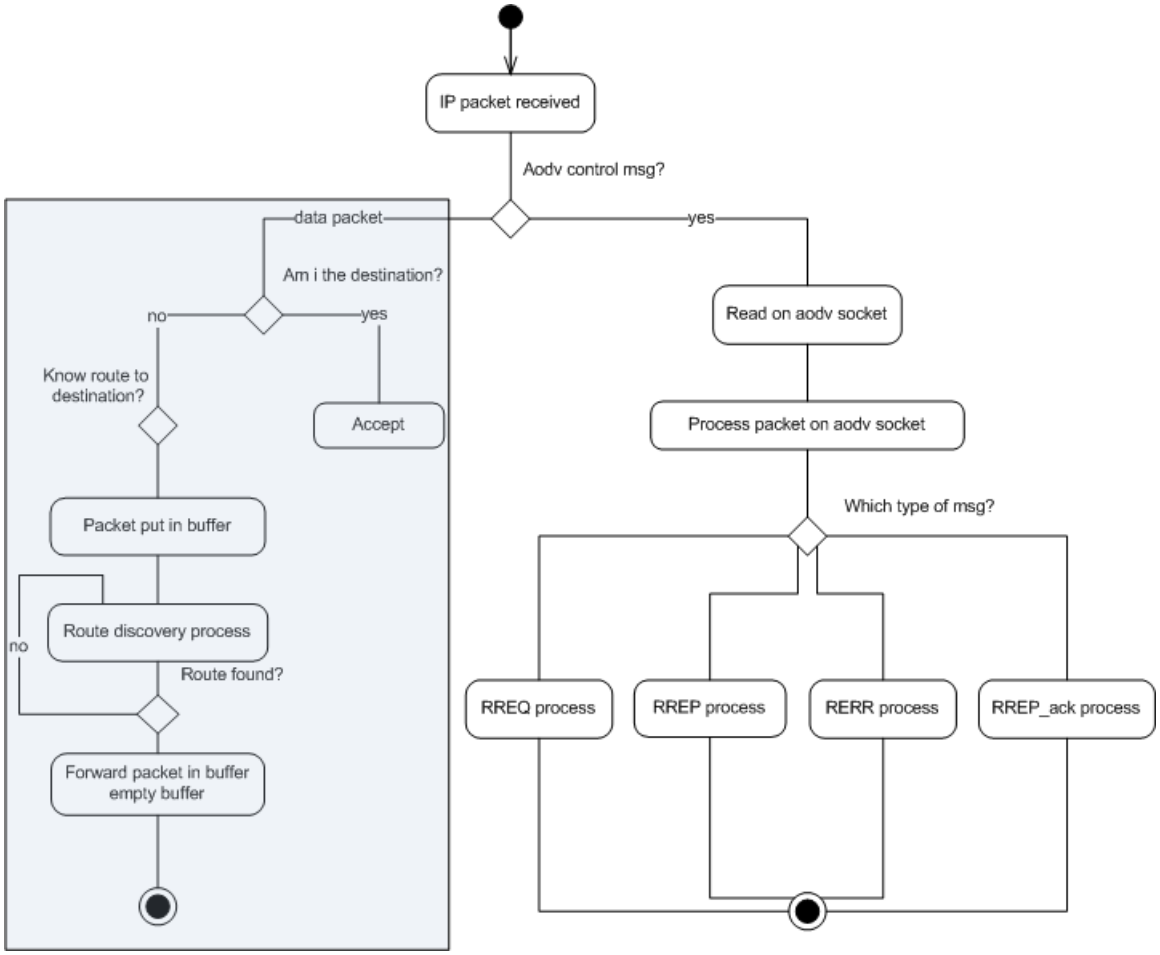



Figure 7: General packet processing

As explained in the previous section, four types of message are defined in AODV-UU, i.e., RREQ, RREP, RERR, RREP-ACK. Since we analyzed the source code through the reputation mechanism, we focused only on the creation and processing of control messages that request a route (RREQ) and on the reply (RREP) to them. Indeed the reputation mechanism does not use the other two types of message.

In the diagram 7, the boxed section on the left in the blue rectangle is not implemented in this way in the source code but the logic remains the same. So, to improve understanding we have decided to simplify this part which normally requires a study of the kernel module used by the Linux kernel. Once an IP packet is received, in case of a data packet, AODV-UU only operates when the host has to forward this message to one of its neighbors for a destination which is still unknown. The path to the destination can already be known if AODV-UU has previously added this route in its routing table.

In the case of a control message, it is treated in accordance with its type (Fig 7).

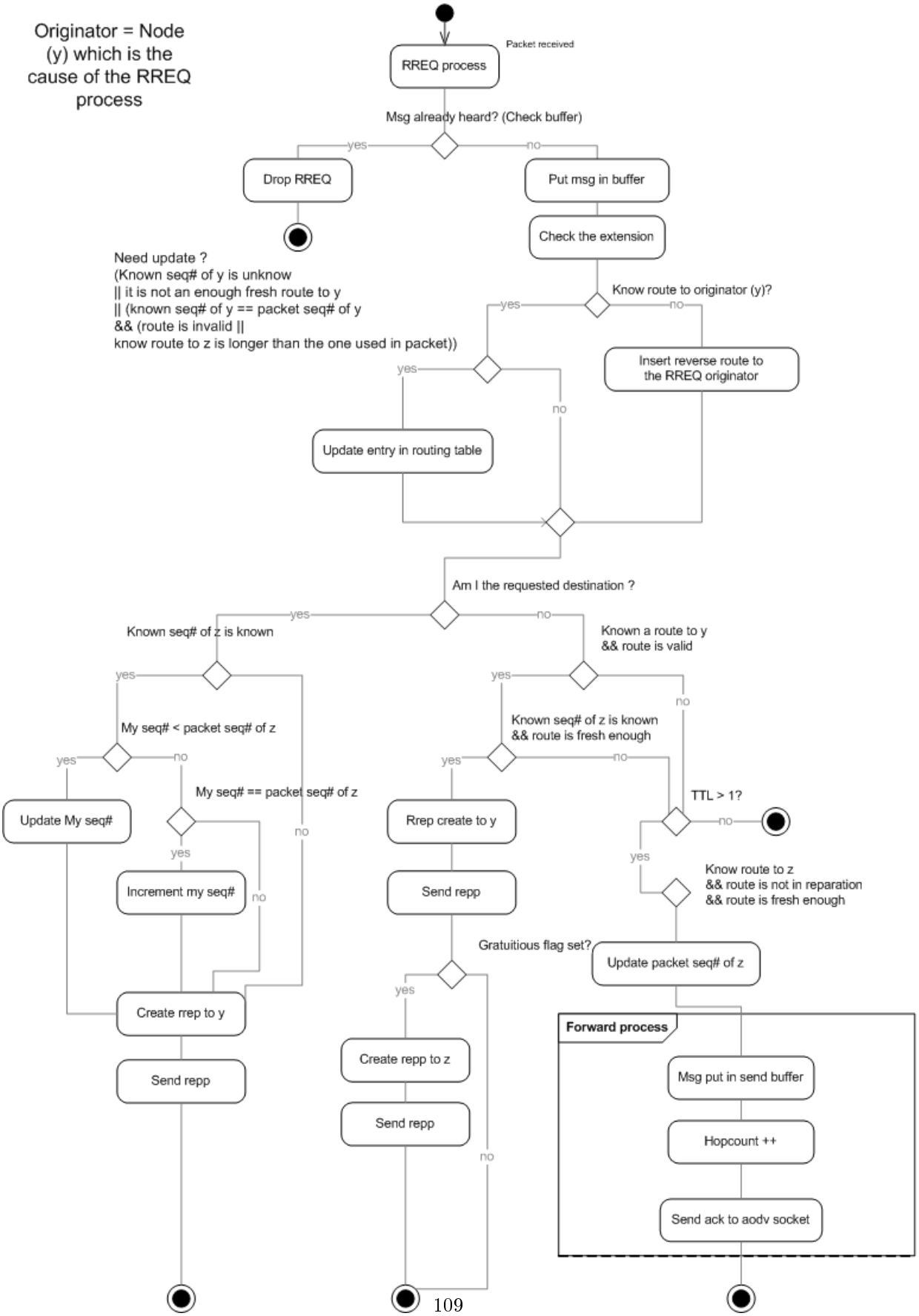

Originator = Node (y) which is the cause of the RREQ process
Packet received
RREQ process
Msg already heard? (Check buffer)
yes
no
Drop RREQ
Put msg in buffer
Check the extension
Need update ?
(Known seq# of y is unknow
|| it is not an enough fresh route to y
|| (known seq# of y == packet seq# of y
&& (route is invalid ||
know route to z is longer than the one used in packet))
Know route to originator (y)?
Insert reverse route to the RREQ originator
Update entry in routing table
Am I the requested destination ?
Known seq# of z is known
Known a route to y && route is valid
My seq# < packet seq# of z
My seq# == packet seq# of z
Update My seq#
Increment my seq#
Create rrep to y
Send repp
Known seq# of z is known && route is fresh enough
Rrep create to y
Send repp
Gratuitious flag set?
Create repp to z
Send repp
TTL > 1?
Know route to z && route is not in reparation && route is fresh enough
Update packet seq# of z
Forward process
Msg put in send buffer
Hopcount ++
Send ack to aodv socket

Figure 8: RREQ processing

When a node receives a RREQ, it first checks if it has not already received it before. If this is the first time, then it adds or updates its entry in the routing table about the originator of the RREQ packet to anticipate the shipment of a likely future RREP message. Then, if the node itself is the destination specified, then it generates a RREP to the originating packet it just received. on the other hand, if not the destination, two specific treatments may be required depending on the knowledge of a valid route to the destination or not (Fig 8).

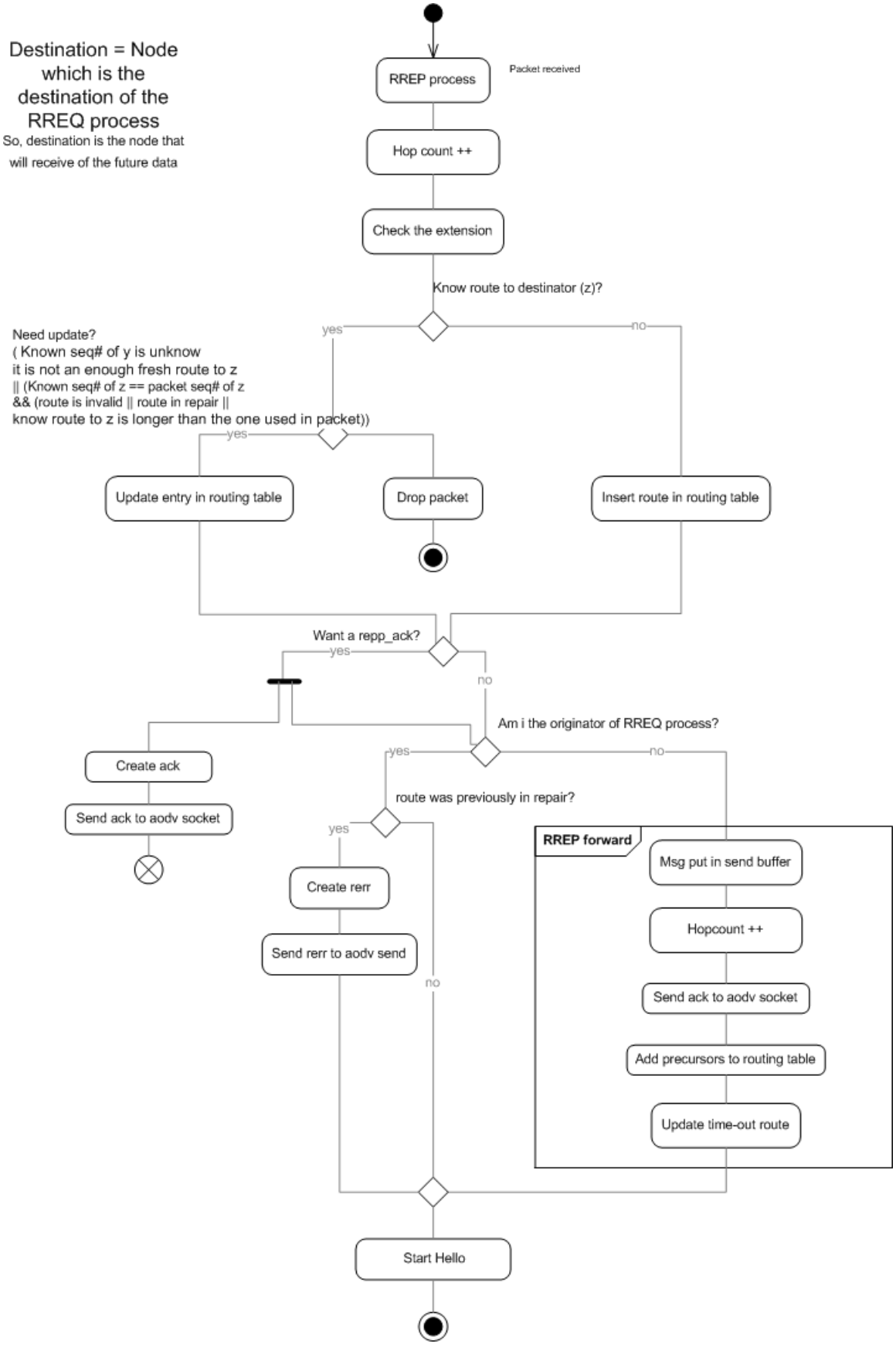

Destination = Node which is the destination of the RREQ process
So, destination is the node that will receive of the future data
RREP process
Packet received
Hop count ++
Check the extension
Know route to destinator (z)?
yes
no
Need update?
( Known seq# of y is unknow
it is not an enough fresh route to z
|| (Known seq# of z == packet seq# of z
&& (route is invalid || route in repair ||
know route to z is longer than the one used in packet))
yes
Update entry in routing table
Drop packet
Insert route in routing table
Want a repp_ack?
yes
no
Create ack
Send ack to aodv socket
Am i the originator of RREQ process?
yes
no
route was previously in repair?
yes
Create rerr
Send rerr to aodv send
no
RREP forward
Msg put in send buffer
Hopcount ++
Send ack to aodv socket
Add precursors to routing table
Update time-out route
Start Hello

Figure 9: RREP processing

Upon receiving a RREP, a first check is performed to ensure that the node has a valid route to the generator of the message. If not, or if the information is outdated, an update is performed. In case the packet target is the current node, the information carried by the RREP are inserted in the routing table. On the other hand, if the current node is not the target but simply an intermediary, it sends the message to one of its neighbors after a possible update of its routing table (Fig 9).

# .3 Main structures used by AODV-FUUREX

For a better understanding of the AODV-FUUREX source code, the main structures will be explained. The various structures shown below were all built using the same basic architecture: a hash table (64 cells) references its elements identified by a network address. An IP address is associated to a single cell of this table and multiple IP addresses can be associated with a cell.

## .3.1 Routing table

Present in the basic version of AODV, these structures contain all the information required to the routing mechanism. AODV-UU maintains a routing table synchronized with the one from the Linux kernel.

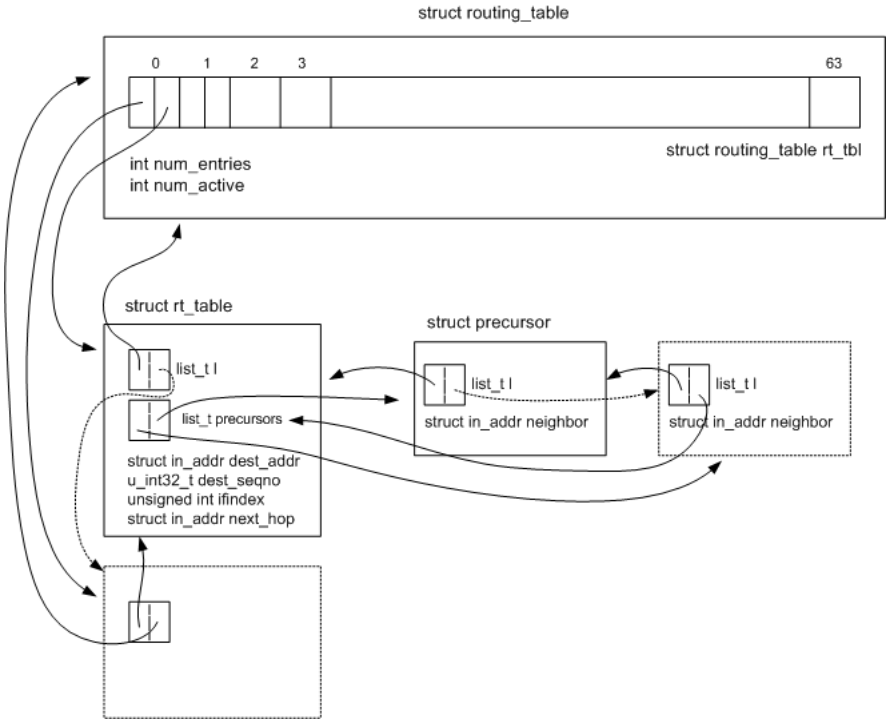



Figure 10: Structures needed by the routing mechanism

## .3.2 Reputation table

The required structures to store the reputation data for the proper execution of the reputation mechanism :

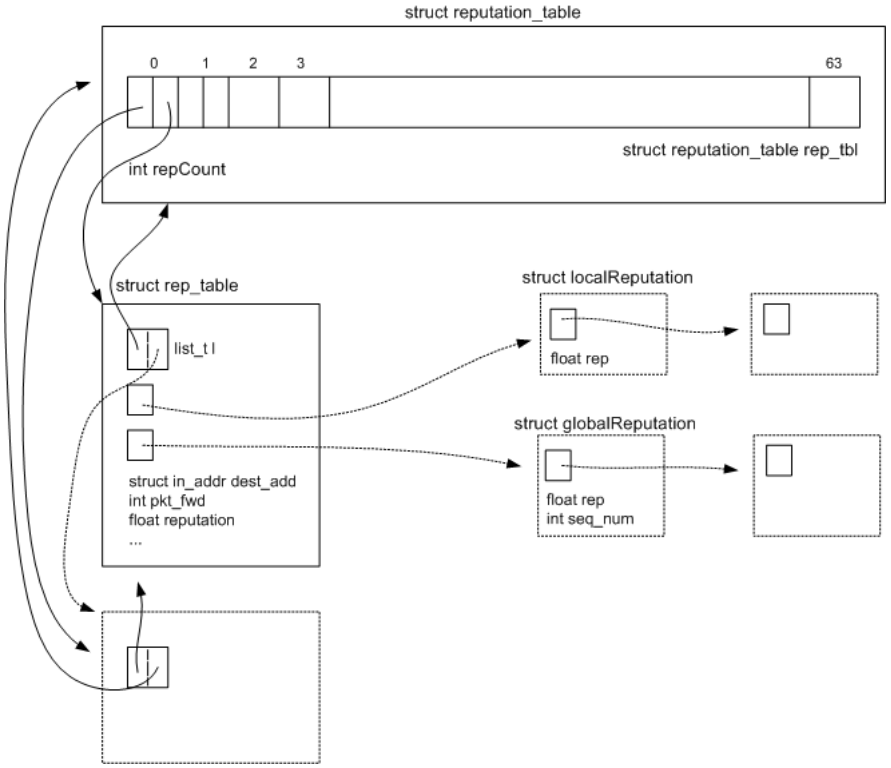



Figure 11: Structures needed by the reputation mechanism

### .3.3 Ack-packets table

These structures can store the packets awaiting acknowledgement by the Watchdog module. They contain all the necessary information to identify a data packet passing through a node whose role is to convey those packets.

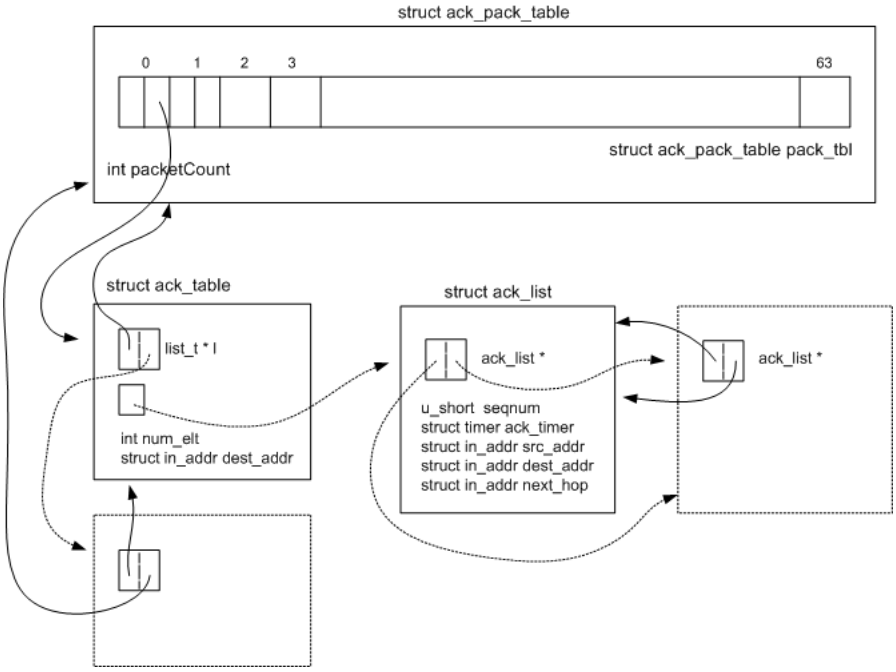



Figure 12: Structures needed by the watchdog mechanism

## .4 AODV-FUUREX Experiment

This experiment enables us to run the experiment to test our implementation on the OMF tesbed. Because some problems could arise during the experiment (we do not control the packet loss on a node that does not work properly), we had to control the actions to be performed in real time. This is the reason why it is not scheduled in the OMF script.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="file:/XMLScenarioDescription/omf.
       xsl"?>
3  <scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlName="
       5nodes3flowfull.xml" xsi:noNamespaceSchemaLocation="..\Documents\
       stage_naples\projet canonico\XMLScenarioDescription\XSD\
       SimulationScenario.xsd">
4    <networkDescription>
5      <description>This is an experiment with AODV</description>
6      <as id="1" name="general AS">
7        <description/>
8        <topologyModel>
9          <description/>
10         <node id="105" x="8" y="105" posX="0" posY="0" nomadicity="0"
               name="Gaia">
11           <label>n2</label>
12           <protocol>
13             <udp id="1"/>
14             <Null id="2"/>
15           </protocol>
16           <cfgNetInterface mode="ad-hoc" type="a" essid="link1" channel="
                 6" ipAddress="10.0.0.105" device="w0"/>
17           <routingDaemon>
18             <aodv>
19               <interface>w0</interface>
20             </aodv>
21           </routingDaemon>
22         </node>
23         <node id="104" x="8" y="104" posX="0" posY="0" nomadicity="0"
               name="Zeus">
24           <label>n1</label>
25           <protocol>
26             <udp id="1"/>
27             <Null id="2"/>
28           </protocol>
29           <cfgNetInterface mode="ad-hoc" type="a" essid="link1" channel="
                 6" ipAddress="10.0.0.104" device="w0"/>
30           <routingDaemon>
31             <aodv>
32               <interface>w0</interface>
33             </aodv>
34           </routingDaemon>
35         </node>
36         <node id="106" x="8" y="106" posX="0" posY="0" nomadicity="0"
               name="Pandore">
37           <label>n3</label>
38           <protocol>
39             <udp id="1"/>
40             <Null id="2"/>
41           </protocol>
42           <cfgNetInterface mode="ad-hoc" type="a" essid="link1" channel="
                 6" ipAddress="10.0.0.106" device="w0"/>
43           <routingDaemon>
44             <aodv>
```

```
45              <interface>w0</interface>
46            </aodv>
47          </routingDaemon>
48        </node>
49        <node id="107" x="8" y="107" posX="0"  posY="0" nomadicity="0"
              name="Aphrodite">
50          <label>n5</label>
51          <protocol>
52            <udp id="1"/>
53            <Null id="2"/>
54          </protocol>
55          <cfgNetInterface mode="ad-hoc" type="a" essid="link1" channel="
                6" ipAddress="10.0.0.107" device="w0"/>
56          <routingDaemon>
57            <aodv>
58              <interface>w0</interface>
59            </aodv>
60          </routingDaemon>
61        </node>
62        <node id="108" x="8" y="108" posX="0" posY="0" nomadicity="0"
              name="Poseidon">
63          <label>n6</label>
64          <protocol>
65            <udp id="2"/>
66            <Null id="1"/>
67          </protocol>
68          <cfgNetInterface mode="ad-hoc" type="a" essid="link1" channel="
                6" ipAddress="10.0.0.108" device="w0"/>
69          <routingDaemon>
70            <aodv>
71              <interface>w0</interface>
72            </aodv>
73          </routingDaemon>
74        </node>
75        <link isFullDuplex="1">
76          <node1 nodeId="105"/>
77          <node2 nodeId="104"/>
78        </link>
79        <link isFullDuplex="1">
80          <node1 nodeId="104"/>
81          <node2 nodeId="106"/>
82        </link>
83        <link isFullDuplex="1">
84          <node1 nodeId="106"/>
85          <node2 nodeId="108"/>
86        </link>
87        <link isFullDuplex="1">
88          <node1 nodeId="105"/>
89          <node2 nodeId="107"/>
90        </link>
91        <link isFullDuplex="1">
92          <node1 nodeId="107"/>
93          <node2 nodeId="108"/>
94        </link>
95      </topologyModel>
96    </as>
97  </networkDescription>
98  <simulationCommand stopTime="240" startTime="0" startLog="0" stopLog="
        238">
99  </simulationCommand>
100 <authors>Julien, Laurent</authors>
```

```
</scenario>
```

## .5 validity of our XML tool

Experiment 1

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="..\XSL\OMF\omf.xsl"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="..\..\XMLScenarioDescription\XSD\
    SimulationScenario.xsd" xmlName="5nodes.xml">
  <networkDescription>
    <description>This is an experiment with OLSR</description>
    <as id="1" name="general AS">
      <description/>
      <topologyModel>
        <description/>
        <node id="105" x="8" y="105" posX="0" posY="0" nomadicity="0"
            name="zeus">
          <label>n1</label>
          <protocol>
            <udp id="1"/>
          </protocol>
          <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
              6" ipAddress="10.0.0.115" device="w1"/>
          <routingDaemon>
            <olsr/>
          </routingDaemon>
        </node>
        <node id="102" x="8" y="102" posX="0" posY="0" nomadicity="0"
            name="keops">
          <label>n2</label>
          <protocol>
            <udp id="1"/>
            <Null id="2"/>
          </protocol>
          <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
              48" ipAddress="10.0.0.102" device="w0"/>
          <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
              6" ipAddress="10.0.0.112" device="w1"/>
          <routingDaemon>
            <olsr/>
          </routingDaemon>
        </node>
        <node id="101" x="8" y="101" posX="0" posY="0" nomadicity="0"
            name="arthemis">
          <label>n3</label>
          <protocol>
            <udp id="1"/>
            <Null id="2"/>
          </protocol>
          <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
              48" ipAddress="10.0.0.101" device="w0"/>
          <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
              1" ipAddress="10.0.0.111" device="w1"/>
          <routingDaemon>
            <olsr/>
          </routingDaemon>
        </node>
```

```
44          <node id="100" x="8" y="100" posX="0" posY="0" nomadicity="0"
                name="hypnos">
45            <label>n5</label>
46            <protocol>
47              <Null id="1"/>
48            </protocol>
49            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.100" device="w0"/>
50            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.110" device="w1"/>
51            <routingDaemon>
52              <olsr/>
53            </routingDaemon>
54          </node>
55          <node id="104" x="8" y="104" posX="0" posY="0" nomadicity="0"
                name="dionysos">
56            <label>n6</label>
57            <protocol>
58              <Null id="1"/>
59            </protocol>
60            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.104" device="w0"/>
61            <routingDaemon>
62              <olsr/>
63            </routingDaemon>
64          </node>
65        </topologyModel>
66      </as>
67    </networkDescription>
68    <traffic>
69      <description>Generated traffic from node 105 to node 104</description
          >
70      <pattern id="1">
71        <src asId="1" nodeId="105" protocolId="1" netInterface="w1"/>
72        <dst asId="1" nodeId="104" protocolId="1" netInterface="w0"/>
73        <trafficModel>
74          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
                packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
                Mbps" randomNoise="false"/>
75        </trafficModel>
76      </pattern>
77    </traffic>
78    <simulationCommand stopTime="60" startTime="0" startLog="0" stopLog="58
        ">
79      <actions>
80        <startTraffic idTraffic="1" at="2"/>
81        <stopTraffic idTraffic="1" at="50"/>
82      </actions>
83    </simulationCommand>
84    <authors>Julien, Laurent</authors>
85  </scenario>
```

Experiment 2

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <?xml-stylesheet type="text/xsl" href="..\XSL\OMF\omf.xsl"?>
3   <scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="..\..\XMLScenarioDescription\XSD\
        SimulationScenario.xsd"  xmlName="5nodes2crossFlows.xml">
4     <networkDescription>
5       <description>This is an experiment with OLSR</description>
```

```
6       <as id="1" name="general AS">
7         <description/>
8         <topologyModel>
9           <description/>
10          <node id="105" posX="0" posY="0" x="8" y="105" nomadicity="0"
                name="zeus">
11            <label>n1</label>
12            <protocol>
13              <udp id="1"/>
14            </protocol>
15            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.115" device="w1"/>
16            <routingDaemon>
17              <olsr/>
18            </routingDaemon>
19          </node>
20          <node id="102" posX="0" posY="0" x="8" y="102" nomadicity="0"
                name="keops">
21            <label>n2</label>
22            <protocol>
23              <Null id="1"/>
24            </protocol>
25            <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
                  48" ipAddress="10.0.0.102" device="w0"/>
26            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.112" device="w1"/>
27            <routingDaemon>
28              <olsr/>
29            </routingDaemon>
30          </node>
31          <node id="101" posX="0" posY="0" x="8" y="101" nomadicity="0"
                name="arthemis">
32            <label>n3</label>
33            <protocol>
34              <Null id="1"/>
35            </protocol>
36            <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
                  48" ipAddress="10.0.0.101" device="w0"/>
37            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.111" device="w1"/>
38            <routingDaemon>
39              <olsr/>
40            </routingDaemon>
41          </node>
42          <node id="100" posX="0" posY="0" x="8" y="100" nomadicity="0"
                name="hypnos">
43            <label>n5</label>
44            <protocol>
45              <udp id="1"/>
46            </protocol>
47            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.100" device="w0"/>
48            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.110" device="w1"/>
49            <routingDaemon>
50              <olsr/>
51            </routingDaemon>
52          </node>
53          <node id="104" posX="0" posY="0" x="8" y="104" nomadicity="0"
                name="dionysos">
54            <label>n6</label>
```

```
            <protocol>
            </protocol>
            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                36" ipAddress="10.0.0.104" device="w0"/>
            <routingDaemon>
              <olsr/>
            </routingDaemon>
          </node>
        </topologyModel>
      </as>
    </networkDescription>
    <traffic>
      <description>Generated traffic from node 105 to node 104</description
          >
      <pattern id="1">
        <src asId="1" nodeId="105" protocolId="1" netInterface="w1"/>
        <dst asId="1" nodeId="101" protocolId="1" netInterface="w0"/>
        <trafficModel>
          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
              packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
              Mbps" randomNoise="false"/>
        </trafficModel>
      </pattern>
      <pattern id="2">
        <src asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
        <dst asId="1" nodeId="102" protocolId="1" netInterface="w0"/>
        <trafficModel>
          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
              packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
              Mbps" randomNoise="false"/>
        </trafficModel>
      </pattern>
    </traffic>
    <simulationCommand stopTime="60" startTime="0" startLog="0" stopLog="58
        ">
      <actions>
        <startTraffic idTraffic="1" at="2"/>
        <startTraffic idTraffic="2" at="2"/>
        <stopTraffic idTraffic="1" at="40"/>
        <stopTraffic idTraffic="2" at="40"/>
      </actions>
    </simulationCommand>
    <authors>Julien, Laurent</authors>
  </scenario>
```

Experiment 3

```
  <?xml version="1.0" encoding="UTF-8"?>
  <?xml-stylesheet type="text/xsl" href="..\XSL\OMF\omf.xsl"?>
  <scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlName="
      5nodes2Proto.xml" xsi:noNamespaceSchemaLocation="..\XSD\
      SimulationScenario.xsd">
    <networkDescription>
      <description>This is an experiment with OLSR</description>
      <as id="1" name="general AS">
        <description/>
        <topologyModel>
          <description/>
          <node id="105" posX="0" posY="0" x="8" y="105" nomadicity="0"
              name="zeus">
            <label>n1</label>
```

```
12            <protocol>
13              <udp id="1"/>
14            </protocol>
15            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.115" device="w1"/>
16            <routingDaemon>
17              <olsr/>
18            </routingDaemon>
19          </node>
20          <node id="102" posX="0" posY="0" x="8" y="102" nomadicity="0"
                name="keops">
21            <label>n2</label>
22            <protocol>
23              <Null id="1"/>
24            </protocol>
25            <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
                  48" ipAddress="10.0.0.102" device="w0"/>
26            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.112" device="w1"/>
27            <routingDaemon>
28              <olsr/>
29            </routingDaemon>
30          </node>
31          <node id="101" posX="0" posY="0" x="8" y="101" nomadicity="0"
                name="arthemis">
32            <label>n3</label>
33            <protocol>
34            </protocol>
35            <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
                  48" ipAddress="10.0.0.101" device="w0"/>
36            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.111" device="w1"/>
37            <routingDaemon>
38              <olsr/>
39            </routingDaemon>
40          </node>
41          <node id="100" posX="0" posY="0" x="8" y="100" nomadicity="0"
                name="hypnos">
42            <label>n5</label>
43            <protocol>
44              <udp id="1"/>
45              <Null id="2"/>
46            </protocol>
47            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.100" device="w0"/>
48            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.110" device="w1"/>
49            <routingDaemon>
50              <olsr/>
51            </routingDaemon>
52          </node>
53          <node id="104" posX="0" posY="0" x="8" y="104" nomadicity="0"
                name="dionysos">
54            <label>n6</label>
55            <protocol>
56            </protocol>
57            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.104" device="w0"/>
58            <routingDaemon>
59              <olsr/>
60            </routingDaemon>
```

```
        </node>
      </topologyModel>
    </as>
  </networkDescription>
  <traffic>
    <description>Generated traffic from node 105 to node 104</description>
    <pattern id="1">
      <src asId="1" nodeId="105" protocolId="1" netInterface="w1"/>
      <dst asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
      <trafficModel>
        <cbr interval="10000" intervalUnit="ms" senderPort="3000"
            packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
            Mbps" randomNoise="false"/>
      </trafficModel>
    </pattern>
    <pattern id="2">
      <src asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
      <dst asId="1" nodeId="102" protocolId="1" netInterface="w0"/>
      <trafficModel>
        <cbr interval="10000" intervalUnit="ms" senderPort="3002"
            packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
            Mbps" randomNoise="false"/>
      </trafficModel>
    </pattern>
  </traffic>
  <simulationCommand stopTime="60" startTime="0" startLog="0" stopLog="58
      ">
    <actions>
      <startTraffic idTraffic="1" at="2"/>
      <startTraffic idTraffic="2" at="2"/>
      <stopTraffic idTraffic="1" at="40"/>
      <stopTraffic idTraffic="2" at="40"/>
    </actions>
  </simulationCommand>
  <authors>Julien, Laurent</authors>
</scenario>
```

Experiment 4

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="..\XSL\OMF\omf.xsl"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlName="
    5nodes3flowfull.xml" xsi:noNamespaceSchemaLocation="..\XSD\
    SimulationScenario.xsd">
  <networkDescription>
    <description>This is an experiment with OLSR</description>
    <as id="1" name="general AS">
      <description/>
      <topologyModel>
        <description/>
        <node id="105" posX="0" posY="0" x="8" y="105" nomadicity="0"
            name="zeus">
          <label>n1</label>
          <protocol>
            <udp id="1"/>
          </protocol>
          <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
              6" ipAddress="10.0.0.115" device="w1"/>
          <routingDaemon>
            <olsr/>
```

```
18            </routingDaemon>
19          </node>
20          <node id="102" posX="0" posY="0" x="8" y="102" nomadicity="0"
                name="keops">
21            <label>n2</label>
22            <protocol>
23              <udp id="1"/>
24              <Null id="2"/>
25            </protocol>
26            <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
                  48" ipAddress="10.0.0.102" device="w0"/>
27            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.112" device="w1"/>
28            <routingDaemon>
29              <olsr/>
30            </routingDaemon>
31          </node>
32          <node id="101" posX="0" posY="0" x="8" y="101" nomadicity="0"
                name="arthemis">
33            <label>n3</label>
34            <protocol>
35              <udp id="1"/>
36            </protocol>
37            <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="
                  48" ipAddress="10.0.0.101" device="w0"/>
38            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.111" device="w1"/>
39            <routingDaemon>
40              <olsr/>
41            </routingDaemon>
42          </node>
43          <node id="100" posX="0" posY="0" x="8" y="100" nomadicity="0"
                name="hypnos">
44            <label>n5</label>
45            <protocol>
46              <Null id="1"/>
47            </protocol>
48            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.100" device="w0"/>
49            <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="
                  1" ipAddress="10.0.0.110" device="w1"/>
50            <routingDaemon>
51              <olsr/>
52            </routingDaemon>
53          </node>
54          <node id="104" posX="0" posY="0" x="8" y="104" nomadicity="0"
                name="dionysos">
55            <label>n6</label>
56            <protocol>
57              <Null id="1"/>
58            </protocol>
59            <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="
                  36" ipAddress="10.0.0.104" device="w0"/>
60            <routingDaemon>
61              <olsr/>
62            </routingDaemon>
63          </node>
64        </topologyModel>
65      </as>
66    </networkDescription>
67    <traffic>
```

```
68      <description>Generated traffic from node 105 to node 104</description
            >
69      <pattern id="1">
70        <src asId="1" nodeId="105" protocolId="1" netInterface="w1"/>
71        <dst asId="1" nodeId="104" protocolId="1" netInterface="w0"/>
72        <trafficModel>
73          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
                packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
                Mbps" randomNoise="false"/>
74        </trafficModel>
75      </pattern>
76      <pattern id="2">
77        <src asId="1" nodeId="101" protocolId="1" netInterface="w0"/>
78        <dst asId="1" nodeId="102" protocolId="1" netInterface="w0"/>
79        <trafficModel>
80          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
                packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
                Mbps" randomNoise="false"/>
81        </trafficModel>
82      </pattern>
83      <pattern id="3">
84        <src asId="1" nodeId="102" protocolId="1" netInterface="w0"/>
85        <dst asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
86        <trafficModel>
87          <cbr interval="10000" intervalUnit="ms" senderPort="3002"
                packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="
                Mbps" randomNoise="false"/>
88        </trafficModel>
89      </pattern>
90    </traffic>
91    <simulationCommand stopTime="60" startTime="0" startLog="0" stopLog="58
          ">
92      <actions>
93        <startTraffic idTraffic="1" at="2"/>
94        <startTraffic idTraffic="2" at="10"/>
95        <startTraffic idTraffic="3" at="10"/>
96        <stopTraffic idTraffic="1" at="40"/>
97        <stopTraffic idTraffic="2" at="40"/>
98        <stopTraffic idTraffic="3" at="40"/>
99      </actions>
100   </simulationCommand>
101   <authors>Julien, Laurent</authors>
102 </scenario>
```

Experiment 5

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="..\XSL\OMF\omf.xsl"?>
3  <scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="..\..\XMLScenarioDescription\XSD\
       SimulationScenario.xsd"  xmlName="5nodesforfun.xml">
4    <networkDescription>
5      <description>This is an experiment with OLSR</description>
6      <as id="1" name="general AS">
7        <description/>
8        <topologyModel>
9          <description/>
10         <node id="105" posX="0" posY="0" x="8" y="105" nomadicity="0"
               name="zeus">
11           <label>n1</label>
12           <protocol>
```

```
13              <udp id="1"/>
14            </protocol>
15            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.115" device="w1"/>
16            <routingDaemon>
17              <olsr/>
18            </routingDaemon>
19          </node>
20          <node id="102" posX="0" posY="0" x="8" y="102" nomadicity="0"
                name="keops">
21            <label>n2</label>
22            <protocol>
23              <udp id="1"/>
24            </protocol>
25            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.112" device="w1"/>
26            <routingDaemon>
27              <olsr/>
28            </routingDaemon>
29          </node>
30          <node id="101" posX="0" posY="0" x="8" y="101" nomadicity="0"
                name="arthemis">
31            <label>n3</label>
32            <protocol>
33              <udp id="1"/>
34            </protocol>
35            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.111" device="w1"/>
36            <routingDaemon>
37              <olsr/>
38            </routingDaemon>
39          </node>
40          <node id="100"  posX="0" posY="0" x="8" y="100" nomadicity="0"
                name="hypnos">
41            <label>n5</label>
42            <protocol>
43              <Null id="1"/>
44              <Null id="2"/>
45              <Null id="3"/>
46              <Null id="4"/>
47            </protocol>
48            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.110" device="w1"/>
49            <routingDaemon>
50              <olsr/>
51            </routingDaemon>
52          </node>
53          <node id="104" posX="0" posY="0" x="8" y="104" nomadicity="0"
                name="dionysos">
54            <label>n6</label>
55            <protocol>
56              <udp id="1"/>
57            </protocol>
58            <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="
                  6" ipAddress="10.0.0.114" device="w1"/>
59            <routingDaemon>
60              <olsr/>
61            </routingDaemon>
62          </node>
63        </topologyModel>
64      </as>
```

```
65    </networkDescription>
66    <traffic>
67      <description>Generated traffic from node 105 to node 104</description
          >
68      <pattern id="1">
69        <src asId="1" nodeId="104" protocolId="1" netInterface="w1"/>
70        <dst asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
71        <trafficModel>
72          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
              destinationPort="4001" packetSize="1024" packetSizeUnit="byte"
               rate="1" rateUnit="Mbps" randomNoise="false"/>
73        </trafficModel>
74      </pattern>
75      <pattern id="2">
76        <src asId="1" nodeId="102" protocolId="1" netInterface="w1"/>
77        <dst asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
78        <trafficModel>
79          <cbr interval="10000" intervalUnit="ms" senderPort="3000"
              destinationPort="4002" packetSize="1024" packetSizeUnit="byte"
               rate="1" rateUnit="Mbps" randomNoise="false"/>
80        </trafficModel>
81      </pattern>
82      <pattern id="3">
83        <src asId="1" nodeId="105" protocolId="1" netInterface="w1"/>
84        <dst asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
85        <trafficModel>
86          <cbr interval="10000" intervalUnit="ms" senderPort="3002"
              destinationPort="4003" packetSize="1024" packetSizeUnit="byte"
               rate="1" rateUnit="Mbps" randomNoise="false"/>
87        </trafficModel>
88      </pattern>
89      <pattern id="4">
90        <src asId="1" nodeId="101" protocolId="1" netInterface="w1"/>
91        <dst asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
92        <trafficModel>
93          <cbr interval="10000" intervalUnit="ms" senderPort="3002"
              destinationPort="4004" packetSize="1024" packetSizeUnit="byte"
               rate="1" rateUnit="Mbps" randomNoise="false"/>
94        </trafficModel>
95      </pattern>
96    </traffic>
97    <simulationCommand stopTime="60" startTime="0" startLog="0" stopLog="58
        ">
98      <actions>
99        <startTraffic idTraffic="1" at="2"/>
100       <startTraffic idTraffic="2" at="2"/>
101       <startTraffic idTraffic="3" at="2"/>
102       <startTraffic idTraffic="4" at="2"/>
103       <stopTraffic idTraffic="1" at="40"/>
104       <stopTraffic idTraffic="2" at="40"/>
105       <stopTraffic idTraffic="4" at="40"/>
106       <stopTraffic idTraffic="4" at="40"/>
107     </actions>
108   </simulationCommand>
109   <authors>Julien, Laurent</authors>
110 </scenario>
```

University of Namur - Belgium
*Dept. of computer science - Network and security pole*
**An Experiment Translation Through Automatic Generation**

{guillala,jvdsype}@student.fundp.ac.be
Extension to the Master thesis

# How to use the XML tool

Extension

August 17, 2010

# Contents

# Introduction

This document details how to fill an XML network experiment description and how to use the XML tool, both of those goals were developped in the scheme to automatically transform an XML network experiment description into a specific-platform document (NS2 and OMF so far).

The first steps of this project were inspired from the work done by Professor Roberto Canonico (Università degli Studi di Napoli - Facoltà di Ingegneria) explained in the article "An XML Description Language for Web-based Network Simulation".

The present document is divided in five main sections. The first section describes the first part of the network experiment description called **network description**. The second and third section, respectively called **traffic** and **simulation command** follow the same division. For each of these sections, a general overview will be followed by a futher explanation. The fourth section describes how to run the XML tool, i.e. what are the Java-based commands to launch the translation from the XML document into a NS2 or OMF file by the way of the XSL. Finally, the document ends up the explanation with a perspective about the XML project.

# 1 Network Description

An already filled section would look like this:

```
<networkDescription>
    <description>This is a sample experiment with OLSR</description>
    <as id="1" name="general AS">
        <description/>
        <topologyModel>
            <description/>
            <node id="105" posX="0" posY="0" x="8" y="105" nomadicity="0" name="zeus">
                <label>n1</label>
                <protocol>
                    <udp id="1"/>
                </protocol>
                <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="6" ipAddress="10.0.0.115" device="wl"/>
                <routingDaemon>
                    <olsr/>
                </routingDaemon>
            </node>
            <node id="102" posX="0" posY="0" x="8" y="102" nomadicity="0" name="keops">
                <label>n2</label>
                <protocol>
                    <Null id="1"/>
                </protocol>
                <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="48" ipAddress="10.0.0.102" device="w0"/>
                <cfgNetInterface mode="ad-hoc" type="g" essid="link1" channel="6" ipAddress="10.0.0.112" device="wl"/>
                <routingDaemon>
                    <olsr/>
                </routingDaemon>
            </node>
            <node id="101" posX="0" posY="0" x="8" y="101" nomadicity="0" name="arthemis">
                <label>n3</label>
                <protocol>
                    <Null id="1"/>
                </protocol>
                <cfgNetInterface mode="ad-hoc" type="a" essid="link2" channel="48" ipAddress="10.0.0.101" device="w0"/>
                <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="1" ipAddress="10.0.0.111" device="wl"/>
                <routingDaemon>
                    <olsr/>
                </routingDaemon>
            </node>
            <node id="100" posX="0" posY="0" x="8" y="100" nomadicity="0" name="hypnos">
                <label>n5</label>
                <protocol>
                    <udp id="1"/>
                </protocol>
                <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="36" ipAddress="10.0.0.100" device="w0"/>
                <cfgNetInterface mode="ad-hoc" type="g" essid="link3" channel="1" ipAddress="10.0.0.110" device="wl"/>
                <routingDaemon>
                    <olsr/>
                </routingDaemon>
            </node>
            <node id="104" posX="0" posY="0" x="8" y="104" nomadicity="0" name="dionysos">
                <label>n6</label>
                <protocol>
                </protocol>
                <cfgNetInterface mode="ad-hoc" type="a" essid="link4" channel="36" ipAddress="10.0.0.104" device="w0"/>
                <routingDaemon>
                    <olsr/>
                </routingDaemon>
            </node>
        </topologyModel>
    </as>
</networkDescription>
```

Figure 1: Network Description section example

This section handles the autonomous system definition as well as the node definition. For each node, its related network parameters will be defined.

The main element, *networkDescription* is the one that will rule the node experiment description.

**Algebraic description** : $networkDescription[\,]^{+}_{?}$
**Attributes**: /

This main network description handler is composed of at least two elements named *description* and *as*. This element compiles the different definitions of AS.

The description explains briefly how the network description section will be organised.

**Algebraic description** : $description[\,]^{+}_{?}$
**Attributes**: /

The autonomous system section defines how nodes will be organised in itself.

**Algebraic description** : $as[id, name]^{+}$
**Attributes**:

- *id* identifies in a unique way the traffic that is going to be defined.
- *name* enables to give a more human-readable name to the AS.

This sub-main element, *as*, is also composed of at least two elements named *description* and *topology-Model*. This element compiles the different definitions of nodes.

The description explains briefly what the topology model section will define.
**Algebraic description** : $description[\,]^{+}_{?}$
**Attributes**: /

The node section defines how the node will be configured.
**Algebraic description** : $topologyModel[\,]^{+}_{?}$
**Attributes**: /

This sub-sub-main element, *topology model* is also composed of at least two elements named *description* and *node*.
This element compiles the different definitions of nodes.
The description explains briefly what the topology model section will define.
**Algebraic description** : $description[\,]^{+}_{?}$
**Attributes**: /

The node section defines how the node will be configured.
**Algebraic description** : $node[id, posX, posY, x, y, nomadicity, name]^{+}$
**Attributes**:

- *id* identifies on a unique way the traffic that going to be defined.
- *posX* is the x-coordinate of the node location in the laboratory (hard-coded).
- *posY* is the y-coordinate of the node location in the laboratory (hard-coded).
- $x$ representes the x-coordinate of the node in the virtual environment.
- $y$ representes the y-coordinate of the node in the virtual environment.
- *nomadicity* defines if the node will move around or not.
  The accepted values so far are **0** [False] and **1** [True].
- *name* enables to give a more human-readable name to the node.

The node has itself some sub elements used to define the network parameters.

The label gives a unique name to the node.
**Algebraic description** : $label[\,]^{+}_{?}$
**Attributes**: /

The protocol defines the one(s) that will be used by the given node.
**Algebraic description** : $protocol[\,]^{+}_{?}$
**Attributes**: /
The accepted sub-elements so far are **tcp** [Sender], **udp** [Sender] and **Null** [Receiver].
*Sub-attributes*: id identifies the kind of flow that will be sent/received and the node role in that traffic.
The cfgNetInterface defines how the network interface will be configured.
**Algebraic description** : $cfgNetInterface[mode, type, essid, channel, ipAddress, device]^{+}$
**Attributes**:

- *mode* determines the mode to set the below given network interface.
  The accepted values are **ad-hoc**, **master** and **managed**.
- *type* determines which type of the 802.11 standard to select.
  The accepted values are **a**, **b**, **g** and **n**
- *essid* identifies the name of the network where data will be exchanged.
- *channel* determines which channel to listen to and send data on.
  The accepted values are **[1-15]**, **[31-48]**, ...
- *ipAddress* determines the IP-address used used for the below network interface.
- *device* determines which network interface has to be used to vehiculate the traffic.
  The accepted values are **w0**, **w1**, ...

The routing protocol defines the one(s) that will be used by the given node to convey the data.
**Algebraic description** : $routingProtocol[\,]^{+}_{?}$
**Attributes**: /
The accepted sub-elements so far are **olsr** and **aodv**.

# 2 Traffic

An already filled section would look like this:

```
<traffic>
    <description>Generated traffic from node 105 to node 104</description>
    <pattern id="1">
        <src asId="1" nodeId="105" protocolId="1" netInterface="w1"/>
        <dst asId="1" nodeId="101" protocolId="1" netInterface="w0"/>
        <trafficModel>
            <cbr interval="10000" intervalUnit="ms" senderPort="3000" packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="Mbps" randomNoise="false"/>
        </trafficModel>
    </pattern>
    <pattern id="2">
        <src asId="1" nodeId="100" protocolId="1" netInterface="w1"/>
        <dst asId="1" nodeId="102" protocolId="1" netInterface="w0"/>
        <trafficModel>
            <cbr interval="10000" intervalUnit="ms" senderPort="3000" packetSize="1024" packetSizeUnit="byte" rate="1" rateUnit="Mbps" randomNoise="false"/>
        </trafficModel>
    </pattern>
</traffic>
```

Figure 2: Traffic section example

This section handles the traffic(s) definition with the description of its elements.

The main element, *traffic*, is the one that will contain and define the different traffics between the nodes.
**Algebraic description** : $traffic[\,]^{+}_{?}$
**Attributes**: /

This main traffic handler is composed of at least two elements named *description* and *pattern*.
This element compiles the different traffics.

The description explains briefly what the traffic section will be doing.
**Algebraic description** : $description[\,]^{+}_{?}$
**Attributes**: /

The pattern defines a flow between two nodes according to a given traffic model.
**Algebraic description** : $pattern[id]^{+}$
**Attributes**:

- *id* identifies in a unique way the traffic that is going to be defined.

This defines what nodes are taking part in the flows, specifying which one will be the sender and which other will be the recipient.
**Algebraic description** : $src/dst[asId, nodeId, protocolId, netInterface]^{+}_{?}$
**Attributes**:

- *asId* references the autonomous system the node is taking part in and where the flow has to be streamed to.
- *nodeId* references the node where the flow will come from/will be received.
- *protocolId* references which protocol among those defined for the node will be used for that traffic definition.
- *netInterface* determines which network interface has to be used to vehiculate the traffic.
  The accepted values are **w0**, **w1**, ...

The traffic model defines what is the type of flows that will be used between nodes.
**Algebraic description** : $trafficModel[\,]^{+}_{?}$
**Attributes**: /

The CBR traffic model is one type of model that can be used to defines flows between nodes.

**Algebraic description** : $cbr[interval, intervalUnit, senderPort, packetSize, packetSizeUnit, rate, rateUnit, randomN$
**Attributes**:

- *interval* determines the interval between the sending of two packet respecting the constant bit rate model.
- *intervalUnit* is the unit of the intervale previously defined.
  The accepted values are **ns**, **ms**, **s**, **h**.
- *senderPort* defines the port at which the sender has to send the traffic.
  In case of multi-flows from and to a single node, it has to define which port will be used to send traffics and where it will receive others.
- *packetSize* defines the size of the packet to be vehiculated.
- *packetSizeUnit* defines the size unit of those packets.
  The accepted values are **byte**, **kbyte**, **Mbyte**.
- *rate* defines the rate of the contant bite rate.
- *rateUnit* defines the unit of this rate.
  The accepted values are **bps**, **kbps**, **Mbps**.
- *randomNoise* defines if random has to be simulated on the links, to simulate real conditions (NS2 exclusive).

# 3 Simulation Command

An already filled section would look like this:

```
<simulationCommand stopTime="60" startTime="0" startLog="0" stopLog="58">
    <actions>
        <startTraffic idTraffic="1" at="2"/>
        <startTraffic idTraffic="2" at="2"/>
        <stopTraffic idTraffic="1" at="40"/>
        <stopTraffic idTraffic="2" at="40"/>
    </actions>
</simulationCommand>
```

Figure 3: Simulation Command section example

This section handles the experiment timing and organizes the events that will occur.
The main element, *simulation command* is the one that will rule the experiment time.

**Algebraic description** : $simulation\ command[startTime, stopTime, startLog, stopLog]\ ?^{+}$
**Attributes**:

- *startTime* determines the time at which the experiment will start.
- *stopTime* is the time when the experiment will stop.
- *startLog* determines the time when the logprocess needs to be started.
  Be sure you start the log **after** the beginning of the experiment.
- *stopLog* is the time at which the log process will be stopped.
  Be sure you stop the log **before** the end of the experiment.

This main time handler is composed of a single element named *actions*. This element compiles the different actions (in a single element) that need to be set up to run the experiment.

**Algebraic description** : $actions[\,]\ ?^{+}$
**Attributes**: /

The actions contain at least a pair of elements : a start and a stop traffic.

**Algebraic description** : $start/stopTraffic[idTraffic, at]\ *^{+}$
**Attributes**:

- *idTraffic* references a traffic which is detailed before in the XML document, 2.
- *at* determines the time when the traffic is started/stopped when the experiment will be launched.

# 4 Translation

Once the XML document is filled based on your objective, you then need to transform this file into a specific platform-dedicated file. To enable the transformation, first download *XSLT Xalan processor* (xalan-j_2_7_1-bin-2jars.tar.gz) on the website from the Apache foundation http://xml.apache.org/xalan-j/downloads.html.

This processor, written in Java, has the big advantage to be multiplatforms and to be free. Moreover, it can be run by command lines, like a script.

Then unzip the file in the directory of your choice. You need to import some libraries in your class path before running the XSLT processor.

- Under Windows, you can create an automatic script which takes 3 arguments:
  1. xml file to transform,
  2. txt file for the output,
  3. xsl file for the transformation rules.

This would look like this:

```
java -classpath D:\xalan-j_2_7_1\xalan.jar;
                D:\xalan-j_2_7_1\xercesImpl.jar;
                D:\xalan-j_2_7_1\xml-apis.jar;
                D:\xalan-j_2_7_1\serializer.jar;
                D:\xalan-j_2_7_1\samples\xalansamples.jar
                        org.apache.xalan.xslt.Process -IN %1 -OUT %2 -XSL %3
```

Under Linux, you can use this line (as a script if desired):

```
java -jar xalan.jar -IN myXML.xml -XSL myXSL.xsl -OUT myResult.xxx
```

# 5 Perspective and future

This work was the very first step in our thesis and our engagement in the development of a tool and later, the implementation of a reputation protocol for the WILE-E testbed. The last step in this orientation would have been the implementation of an XML GUI, so the use of the tool would have been made easier.

Besides, a rough outline has been designed but not completed. The GUI is based on the *CASTADIVA* GUI, and rearranged to include all the elements required to perform an experiment on the OMF platform and on the NS2 simulator (for now). The sources can be found at "# # #".

However, this graphic interface would also make it possible to determine the logical position for the NS2 platform. Indeed, the current node location is hard coded, and need to be changed manually. The reason for this is the limited complex functions that can be designed in the XSL language. The position has to be calculated commensurate with the wave scope and the communications between nodes which has been set. This need of correction to amend the current situation has been started.

# Bibliography


[Alt10] Altova. Xmlspy. http://www.altova.com/download.html, Last update: [9 Apr. 2010].

[AvD10] J.Visser A. van Deursen, P. Klint. Domain-specific languages: An annotated bibliography. http://homepages.cwi.nl/~arie/papers/dslbib/, Last update [28 Mar. 2010].

[AY06] T.R. Audel and A. Yasinsac. On the credibility of manet simulations. *Computer, IEEE Computer society*, July 2006.

[CC07] R.A. Calvo and J.P. Campo. Adding multiple interface support in ns2. *University of Cantabria*, January 2007.

[cMFO10] cOntrol and J. Tsai Management Framework (OMF). Case study 3. http://omf.mytestbed.net/wiki/omf/CaseStudy3, Last update: [9 Apr. 2010].

[cO09a] cOntrol and Management Framework (OMF). Glossary. http://omf.mytestbed.net/wiki/omf/Glossary, Last update [20 Dec. 2009].

[cO09b] cOntrol and Management Framework (OMF). Introduction. http://omf.mytestbed.net/wiki/omf/Introduction, Last update [20 Dec. 2009].

[cO09c] cOntrol and Management Framework (OMF). Omf inventory schema. http://omf.mytestbed.net/wiki/1/InstallationInventory_, Last update [20 Dec. 2009].

[cO09d] cOntrol and Management Framework (OMF). User documentation. http://omf.mytestbed.net/wiki/omf/Documentation, Last update [20 Dec. 2009].

[cO10a] cOntrol and Management Framework (OMF). Basic hellow world. http://mytestbed.net/wiki/omf/Basic_Tutorial_Hello_World, Last update: [11 Apr. 2010].

[cO10b] cOntrol and Management Framework (OMF). addlink feature. http://omf.mytestbed.net/wiki/omf/DefTopology, Last update: [9 Apr. 2010].

[cO10c] cOntrol and Management Framework (OMF). Installation inventory. http://omf.mytestbed.net/wiki/1/InstallationInventory, Last update: [9 Apr. 2010].

[Con10] Dan Connolly. Xml tutorial, w3c. http://www.w3schools.com/xml/default.asp, Last update: [10 Apr. 2010].

[DHR] A.Leonhardi D. Herrscher and K. Rothermel. Modeling computer networks for emulation. *Institute of Parallel and Distributed High-Performance Systems (IPVR) - University of Stuttgart.*

[Ecl10] Eclipse. Graphical modeling framework (gmf). http://www.eclipse.org/modeling/gmf/, Last update: [9 Apr. 2010].

[Emu09] EmuLab. Presentation. http://www.emulab.net, Last update [20 Dec. 2009].

[Gar10] By: Lars Marious Garshol. Extended backus-naur form (ebnf). http://www.garshol.priv.no/download/text/bnf.html, Last update: [10 Apr. 2010].

[Gro03] Network Working Group. Ad hoc on-demand distance vector (aodv) routing. http://www.faqs.org/rfcs/rfc3561.html, 2003.

[Hee10] D.van Heesch. Doxygen hompage. http://www.stack.nl/~dimitri/doxygen/, Last update: [24 Apr. 2010].

[HP04] Yih-Chun Hu and Adrian Perrig. A survey of secure wireless ad hoc routing. *IEEE Security and Privacy*, June 2004.

[HWP09] Andrew Hallagan, Bryan Ward, and L. Felipe Perrone. An experiment automation framework for ns-3. *Department of Computer Science - Bucknell University*, 2009.

[IFAab04] Xudong Wang b Ian F. Akyildiz a and Weilin Wang b. Wireless mesh networks: a survey. *Department of Electrical and Computer Engineering*, November 2004.

[JHM09] Carloas T. Calafate Jorge Hortelano, Juan-Carlos Cano and Pietro Manzoni. testing applications in manet environments through emulation. *Technical University of Valencia, Department of Computing Engineering*, 2009.

[JS] Jangeun Jun and Mihail L. Sichitiu. The nominal capacity of wireless mesh networks. *Department of Electrical and Computer Engineering.*

[Kid02] C. Kiddle. The anml guide. *Department of Computer Science) - University of Calgary*, 12 2002.

[net10] netfilter.org. Iptables. http://www.netfilter.org/, Last update: [5 August 2010].

[Nor10a] E. Nordstrom. Aodv-uu homepage. http://core.it.uu.se/core/index.php/AODV-UU/, Last update: [22 Apr. 2010].

[Nor10b] E. Nordstrom. Sourceforge aodv-uu download. http://sourceforge.net/projects/aodvuu/, Last update: [22 Apr. 2010].

[OHH02] J. Schmitt O. Heckmann, K. Pandit and M. Hoffmann. Milestone 2 letsqos scenario generation. *LETSQoS*, 2002.

[OL09] Orbit-Lab. Publications. http://www.orbit-lab.org/wiki/Orbit/Documentation/Publications, Last update [20 Dec. 2009].

[Oli07] F. Oliviero. *On the Effective Exploitation of Distributed Information for Cooperative Network Security and Routing Optimization.* PhD thesis, Universita degli Studi di Napoli Federico II - Facolta di Ingegneria, 2007.

[OLS10] Optimized link state routing protocol. http://www.olsr.org, Last update: [9 Apr. 2010].

[OR07] Francesco Oliviero and Simon Pietro Romano. A reputation-based metric for secure routing in wireless mesh networks. *University of Naples*, 2007.

[oS10] Michael Kay of Saxonica. Saxon, the xslt and xquery processor. http://saxon.sourceforge.net/, Last update: [9 Apr. 2010].

[Pla09] PlanetLab. Presentation. http://www.planet-lab.org, Last update [20 Dec. 2009].

[RCG03] D.Emma R. Canonico and G.Ventre. An xml description language for web-based network simulation. 2003.

[Rub10] Ruby. Ruby official website. http://www.ruby-lang.org/en, Last update [28 Mar. 2010].

[Uni] Cornell University. Cia. http://www.law.cornell.edu/uscode/44/3542.html.

[Wei10] M. Weiser. Feature diagrams. http://www.oonumerics.org/tmpw00/weiser/node8.html, Last update: [21 May 2010].

[xal10] The apache xalan project. http://xml.apache.org/xalan-j/, Last update: [9 Apr. 2010].

[XML10] Xml-nodeset. http://www.xml.com/pub/a/2003/07/16/nodeset.html, Last update: [9 Apr. 2010].